SPARK Rapid Prototyping Environment – Mobile Phone Development Made Easy

Robert Adelmann[1] and Marc Langheinrich[2]

[1] ETH Zurich, Clausiusstrasse 59, 8092 Zurich, Switzerland
adelmann@inf.ethz.ch
[2] University of Lugano (USI), Via Giuseppe Buffi 13, 6904 Lugano, Switzerland
langheinrich@acm.org

Abstract. Over the past few years mobile phones have evolved into attractive platforms for novel types of applications. However, compared to the design and prototyping of desktop software, mobile phone development still requires programmers to have a high level of expertise in both phone architectures and their low-level programming languages. In this paper we analyze common difficulties in mobile phone programming and present SPARK, a publicly available rapid prototyping platform that allows programmers without prior mobile programming experience to create advanced mobile phone applications in a fast and easy way. SPARK currently supports Symbian S60 devices and enables developers to quickly design, test, upload, monitor, and update their applications. We also present the results of a case study, where 70+ students used SPARK to develop mobile applications as part of a graduate course on distributed systems.

Keywords: Rapid prototyping, mobile phone, toolkit, PyS60

1 Introduction

Mobile phones are increasingly becoming a very attractive application platform: They are ubiquitous, highly mobile, provide significant computing power, and often offer an abundance of built-in sensors. Especially in the field of pervasive computing, these devices show a lot of potential when it comes to bridging the often described gap between the real and the virtual word. Concrete projects use mobile camera phones to recognize 1D or 2D bar codes that link information to retail products [1], use the phone's built-in GSM, GPS or WLAN modules for location based services [2], do "reality mining" through Bluetooth sightings in order to map social networks and everyday activities [3], or to generate noise-maps of environments with the help of the built-in microphone [21].

However, despite today's abundance of feature-rich mobile phone hardware and powerful software platforms, creating applications that leverage the platforms' potential is still a time consuming process that challenges non-expert developers by requiring in-depth know-how [19]. This is especially true for applications that require full control of the device and its sensors. The limited API of JavaME [6], the Java

runtime available on many mobile phones, severely restricts in-depth phone control, which forces many application designers to delve into low-level programming languages such as C++ Symbian [7] (used on the majority of today's smart phones) or Objective C [26] (used on the iPhone). While a number of scripting languages are available for mobile phones, there is usually a drawback involved. Either they are still in early development (Lua [27], Ruby [28]), run on top of the phone's Java runtime and thus fair no better than Java ME (Hecl [30]) or allow only limited device control (FlashLite [29]).

One notable exception is the Nokia-initiated Python for S60 (PyS60) [10][4], as it is implemented in native Symbian C++, officially supported by Nokia, offers direct access to most available phone functions, and is extensible through C++ Symbian modules. However, PyS60 development is also not without difficulties, since its Symbian heritage requires programmers to be comfortable with the typical Symbian development process, e.g., how to package an application for distribution, or how to sign an application in order to gain access to sensors like GPS (which includes understanding the complex *Symbian Signed* [5] process and obtaining the appropriate certificates). Last but not least, any application development on mobile phones – be it low-level, Java-based, or scripted – faces three additional challenges when it comes to general development issues: a) since programming is typically done on a desktop or laptop computer, developers must repeatedly upload, debug, and update their software on the actual devices – a process that is often time consuming and fraught with errors; b) programmers must ensure that the software runs on different device types, and c) developers might need to update already deployed applications, e.g. for bug-fixing.

The goal of our system is to provide an easy and fast development environment for mobile phone applications, in particular for developers not familiar with mobile phone programming. This is achieved by providing a rapid prototyping environment that leverages the strengths of the existing PyS60 eco system and systematically addresses the remaining problems of mobile phone prototype development. Section 2 first describes the remaining difficulties of the PyS60 development process in particular and of application design for mobile phones in general. Drawing from these challenges, section 3 then presents the SPARK architecture and its implementation. Section 4 compares SPARK with existing solutions and discusses alternative application development options. SPARK is already used in a number of our research projects, in teaching, and in several larger projects with industry partners. Section 5 discusses an example case studies – a 70+ student class on distributed systems.

2 PyS60 Application Development

Based on several PyS60 projects conducted in the past, we identified a number of general areas in which application development for mobile phones challenges the design and implementation process. These areas are especially problematic for beginners, who face a steep and frustrating learning curve even when using PyS60 instead of Symbian C++. More general issues arise in application scenarios that target large user deployments and frequent code updates, e.g., as part of an early adopter rollout or during long-term user studies.

2.1 Beginner's Issues

When teaching colleagues and students on how to use PyS60, we noted three main obstacles for beginners: application signing, application packaging, and setting up a working development environment:

Application Signing: With Symbian OS v9, Symbian introduced a security system that is based on application *capabilities*[1] and certificates [5]. Getting access to certain features (for example the GPS module or GSM cell information) requires users to choose or compile the Python interpreter and PyS60 modules with the required capabilities and sign them with a certificate covering these capabilities. Getting used to the complex Symbian Signed processes, obtaining software modules that have been compiled with the appropriate capabilities as well as obtaining certificates that grant more capabilities is a time consuming and complex task.

Application Packaging: In order to distribute an application on more than one device, all application files have to be packaged as an SIS[2] (Symbian Installation System) file. This not only requires the above mentioned application signing, but also choosing a matching application identifier, creating the SIS file, and providing an appropriate application icon in a special SVGT format.

Development Environment Setup: Compared to other programming alternatives (especially C++ Symbian), PyS60 drastically simplifies the process of programming. However, even though PyS60 comes with a range of tools to support this process, getting to know what components and tools are required and available for what kind of Symbian version (e.g., there are significant differences between the 2nd and 3rd edition), as well as setting them up, is non-trivial and may differ across desktop operating systems. Preparing, e.g., a homework assignment involving PyS60 that covers all possible combinations of student hardware and software is still prohibitively time consuming.

2.1 General Issues

Even after users became familiar with PyS60 programming, three general issues continued to make the application development process tedious: the need for on-device testing, the variety of available devices, and managing software updates:

On-Device Testing: Emulators are no substitution for on-device testing of applications. They do not behave exactly like real hardware (e.g. regarding stack size, handling of low-memory situation, or timing), nor do they support all features (e.g. camera support). This requires the developer to ultimately test the application on actual devices, including the time consuming task of copying and executing all updated application files on the phone, resulting in long and tedious "code, edit, execute"-cycles.

---

[1] Capabilities control access to various sensitive features of the phone (17 in total), e.g., the current location of the user. If a developer wants a PyS60 program to query the current location, its calling shell must have been registered with this "capability" and subsequently digitally signed with a capability-enabling certificate.

[2] SIS files package application files for installation.

Fig. 1. Multiple mobile phones with the SPARK client software can be connected to a SPARK environment instance running on a PC.

Multi-Device Support: Despite common programming platforms (such as Symbian S60 or even Java ME) every phone model has its differences. Mobile phone applications therefore need to be tested practically on every single device (including different firmware revisions) that they should be deployed on, in order to ensure proper operation. Given the above-mentioned lengthy "code, edit, execute" cycles, testing each program modification on all supported devices is a time consuming task.

Remote Application Updates: Large-scale and/or long-term application deployments invariably come to the point when critical updates or missing features need to be rolled-out to all handsets. A deployment such as the MEA application [11] used in the Metro future store, which allow users to scan products with their phone in order to perform self-checkout, has 100 registered users and around 30 in-store devices – fixing a bug or adding new features would require all users to bring in their mobile phone for servicing, as a typical user would not be able to easily install a provided SIS file on their mobile phone themselves. Developers would be equally challenged by manually updating over 100 devices with a new software version.

3 Rapid Prototyping with SPARK

In the following sections we will present the SPARK architecture and describe its implementation. SPARK specifically focuses on removing the above mentioned obstacles to PyS60 programming, in order to lower the barrier of entry for programmers that are unfamiliar with mobile phone architectures, and to simplify the general development cycles in order to support the development and deployment of large-scale, real-world applications.

The SPARK rapid prototyping environment is an OS independent software that facilitates the easy and fast creation, testing, deployment, and maintenance of arbitrary PyS60 applications. It builds upon the available PyS60 resources [4] and extends these with features that address the six previously identified problems (section 2). The environment comes in an easy to install single package and requires no prior knowledge about mobile phone programming or PyS60. With only a basic knowledge of Python, developers can start creating applications within minutes.

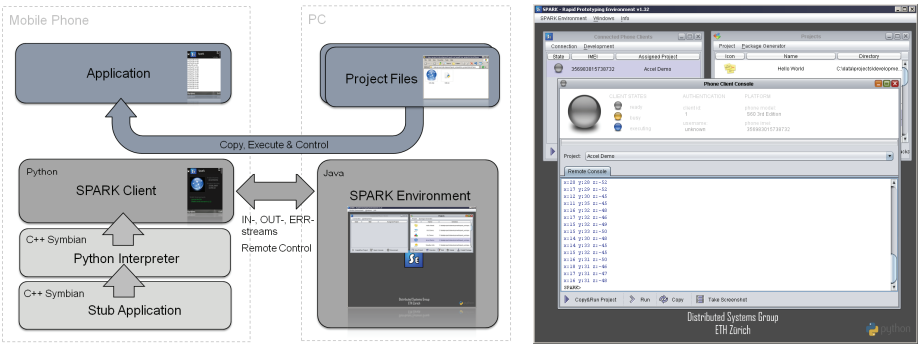


Fig. 2. Left: System architecture. Right: Screenshot of the SPARK environment with three open windows: The left showing all currently connected mobile phones, the right containing the list of projects and the middle being the remote control window for a connected phone showing the current program output.)

Specifically, the design of SPARK followed four basic principles:

- **Low entry barrier:** Application creation should require no prior knowledge about mobile phone programming, and should take a minimal amount of time.
- **No application restrictions:** There should be no restrictions placed on the type of applications that can be created.
- **Fast application creation:** Setup should be fast, "code, edit, execute"-cycles should be quick, and application deployment simple.
- **Ease of use:** Non-expert developers should be able to create powerful applications by abstracting from complex tasks.

The SPARK system consists of two parts: A mobile phone application (SPARK client) as well as a desktop Java application (SPARK environment). Each mobile phone application is represented as a *project* in the SPARK environment. A project is simply a directory on the user's PC that contains all application files. These typically include one or more Python source code files and additional resources like images.

The SPARK *environment* allows users to manage (create, copy, delete) projects, to install and execute projects on connected phones, and to package projects into distributable SIS files. The SPARK *clients* act as containers on the mobile phones that provide a connected SPARK environment with remote access to the device. Multiple SPARK clients (and therefore multiple mobile phones) can be connected to a single SPARK environment (cf. Fig. 1). SPARK supports USB, Bluetooth, WiFi, and GPRS/UMTS connections between an SPARK environment and its clients.

In order to allow novel users to start quickly with development, installation has been designed to be very simple: A prepackaged SIS file must be installed on each development phone, and a single installer file (e.g., EXE for PCs, or an OS independent JAR) needs to be installed on the desktop computer. The SPARK client's SIS file contains all necessary resources needed for PyS60 development on Symbian9 3rd edition smart phones (e.g., the Nokia N73, N95/8g, or business phones like the E61)[3] and thus requires no other software to be present.

[3] See www.s60.com/life/s60phones/browseDevices.do for a full list of devices

3.1 SPARK Environment

The SPARK environment provides several services that apply to the currently connected mobile phones (SPARK clients) and the list of managed projects.

For each connected phone, developers can open a window providing remote access to this device. A remote control window displays basic information about the phone (model, IMEI[4] number), provides a remote Python console with syntax highlighting, and allows the execution of all of the installed projects on the phone by simply pressing a button. The remote console provides developers with a very direct and interactive way to explore PyS60, to test code found on the web (by simply pasting it into the console), or for inspecting parts of their own application.

If multiple (potentially different) mobile phones are connected to the SPARK environment, developers are able to execute projects in parallel on multiple selected phones, and can also monitor their output in real-time, which allows for the easy detection and correction of application problems occurring only on some phone models.

Creating new projects is supported by a wizard dialog, allowing users for example to use application templates as a starting point for own applications. Available templates can easily be changed and extended with user generated ones.

Project management in the SPARK environment is very lightweight. There is no internal Python editor or file manager for managing project files – developers can choose their preferred editor and file manager instead. The SPARK environment simply keeps track of the main project directories, offering various actions for each of them. A typical "code, edit, test"-cycle is as follows: The user makes some changes to one or more project files (e.g., code or images). After saving the changes, she presses the "Copy & Run" button on the SPARK remote console window. The SPARK environment will: determine what files have changed; copy only the changed files to the mobile phone; and then run the updated project directly on the phone through the SPARK client. The application's output can then be monitored on the remote console. This allows for very agile development, as saving changes with "Ctrl-S" and then pressing the "Copy & Run"-button usually requires less than a second.

Once an application is ready to be deployed (i.e., installed as a standalone application), the "application packaging" dialog (see Fig. 3) provides a very simple way of doing this. It allows the generation of SIS files that contain all necessary project files, plus any additional software that might be required for the application to run, e.g., the Python interpreter or any of the third party C++ extension modules shipping with SPARK.

One important feature of SPARK is its auto-update mechanism, as this can significantly help in large-scale deployments. Developers can configure the generated standalone mobile phone applications in such a way that they will periodically monitor for changes in their original project files, and automatically update themselves without the need for end-user interaction. This makes changes to already deployed applications very easy. The "application packaging" dialog allows for the configuration of "When?" and "How?" this updating happens.

---

[4] IMEI, the "International Mobile Equipment Identity", is a worldwide unique number for practically every mobile handset.

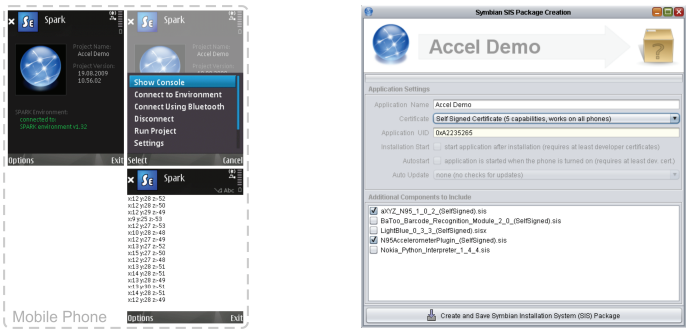


Fig. 3. Screenshots from the SPARK client application and the SIS creation dialog.

3.2 SPARK Client

The SPARK client software allows the SPARK environment to remotely access a device. It also acts as a container for projects that should be executed on the mobile phone. It is a stand-alone application written in PyS60.

Fig. 3 on the left shows the main screen, the options dialog and console view displaying the current program's output. The main screen lists information about the contained projects (name, icon, and versioning information) and indicates whether the SPARK client is currently connected to an SPARK environment instance. Connections to an environment can be established using Bluetooth, WiFi, or the UMTS/GSM network. While WiFi and UMTS connections tend to be very responsive, we have found Bluetooth to provide the shortest delays when working with the interactive console. STDOUT and STDERR messages from executed applications can be viewed not only remotely on an SPARK environment, but also locally (i.e., on the phone) on a console. Once a project has been copied to an SPARK client, the project can also be started directly, without the need for a connected SPARK environment.

4 Related Work

A number of projects have recognized the need for providing rapid prototyping tools for mobile phone development. In this section we will cover prior work on rapid prototyping platforms for mobile phones (section 4.1), discuss alternative options for mobile phone programming (section 4.2), and contrast SPARK with existing tools that specifically target PyS60 programming (section 4.3).

4.1 Rapid Prototyping Platforms

The MakeIt [13] system by Holleis and Schmidt allows for the creation of application interfaces for mobile phones using (state, action)-graphs. MakeIt specifically targets the gap between IDEs and paper prototyping. One of its key features is the fact that

created applications can be simulated and analyzed according to the KLM model. The main difference to SPARK is the fact that the system's focus is on interface creation, and that it allows only for the creation of Java ME templates. This requires developers to additionally use a Java ME development environment and limits the prototypes' functionality to what the Java ME supports.

Campaignr [14] is a C++ Symbian software for mobile phones that supports data collection from a larger set of sensors. It can be used to get both continuous and triggered audio, video and location information from the device, allowing also non experts in C++ Symbian to easily access and use this information. In a similar manner, the MyExperience [31] system also supports the collection of data from users, with the specific feature of allowing users to enrich the data gathered with subjective information that focuses on the user's activities, e.g. "working" or "biking". Compared to SPARK, both Campaignr and MyExperience are specific applications that can be configured using XML files. They are specialized to create customized data-collection applications and do not support other application types.

The ActivityDesigner [32] by Li and Landy allows users to create activity driven prototypes using a visual designer, and to test and deploy these applications on multiple devices, including mobile phones. This platform is similar to SPARK in so far as it targets non-experts, and that it allows for the very simple and fast creation of applications suitable for prolonged real-world usage. However, ActivityDesigner focuses much more on the design process, and only supports mobile phone deployment as part of a JavaScript-based application that runs within a phone's browser.

ContextPhone [15] is a prototyping platform consisting of C++ Symbian modules that can be customized to sense and store data on a mobile phone, and to communicate data between a phone and a "sink" application. While ContextPhone can significantly ease and accelerate application creation for mobile sensing applications, it is not an "out-of-the-box" solution, but a collection of C++ Symbian components that require users to be familiar with C++ Symbian development, which renders it not usable for non-expert developers [19]. The same holds for the rapid application development system presented in [17] that provides a framework for C++ Symbian development, which eases GUI creation, data access and communication.

4.2 Mobile Phone Programming Options

Today's smartphones support an abundance of different programming options. We focused our work on the Symbian S60 platform, as it is still the most prominent platform by far [22], with the largest number of available devices. While the iPhone has recently attracted significant interest and momentum as an open development environment, it requires the use of Objective C. While certainly a powerful language, Objective C is hardly suitable for prototype creation by non-experts. Scripting languages, on the other hand, are currently not supported on the iPhone due to license restrictions from Apple. There are also additional restrictions that make prototype development on iPhones difficult: It supports no background processes, access to the camera's video images is restricted, and the development requires Apple hardware.

Java ME [6] is in general a good programming option for non-experts, featuring extensive tool support and documentation. There is a large set of frameworks that

simplify and accelerate application development, e.g., J2ME Polish [23]. The BaToo toolkit [18] falls into the same category, being mainly a Java ME framework that supports the creation of services to retail products, based on 1D barcode recognition. However, as we pointed out in the introduction, Java ME offers only a very limited set of APIs, and the lack of extension options like the Java Native Interface (JNI) in MIDP [6] severely restricts application capabilities. Examples for restrictions include the lack of support for newer sensors like 3D accelerometers, lack of support for accessing video images from the camera for image recognition tasks, allowing only access to single images, or lack of control of Bluetooth scanning processes.

C++ Symbian offers speed as well as full control of devices, but it features a very steep learning curve, mainly due to the sub-optimal tool support and lack of documentation. Also, the unique Symbian language concepts like ActiveObjects, Descriptors, or Resource files are very complicated to use [19]. For the envisioned fast prototype creation of non-expert developers, C++ Symbian is not really an option.

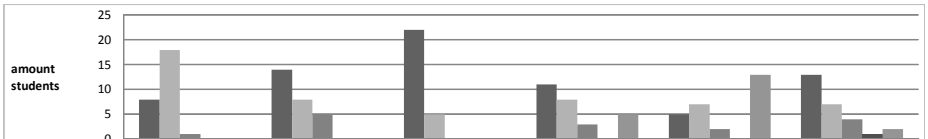
4.1 PyS60 Related Tools

Python for S60 has been introduced by Nokia in 2006 and has since then gained a lot of attention by developers, which is due to several reasons: It is a very easy to learn and use, it features a large set of APIs (especially compared to Java ME), it is extensible through C++ modules, it features an open source license, and there is already an abundance of demo applications for numerous tasks available [24][25].

Due to the popularity of PyS60 there are also tools available that address some of the shortcomings that we identified in section 3.1 above. PUTools [8], as well as the PythonShell application contained in the standard PyS60 Python distribution [4], feature a remote Bluetooth console that allows users to remotely execute commands on the mobile phone. The SPARK implementation of this feature offers an easier setup by completely encapsulating Bluetooth connection setup – a process that needs to be done manually by the developer when using PUTools or the PythonShell. SPARK also supports WiFi and GSM/UMTS for remote connections, e.g. when the desktop machine has no Bluetooth hardware, or when access to the phone is required in “the field”. The latter feature is complemented by allowing SPARK developers, e.g., to inspect devices by taking remote screenshots. Additionally, SPARK provides syntax highlighting and persistent logging, and the Bluetooth console is integrated into the remote console dialog described in section 3.2, offering further features.

PUTools also provides a command line tool for copying files to the mobile phone. SPARK in contrast offers this feature by seamlessly integrating it into the SPARK environment, allowing the developer to copy and execute project files using a single button press. SPARK also simplifies application execution, which is not covered at all by PUTools. Whenever a new version is uploaded to the phone, SPARK automatically ensures that the old module version is unloaded so that the new version will be considered upon execution. Using SPARK, the developer is also relieved from the micro management of where to store her application during development and deployment. SPARK ensures that all Python source code files and resources are always found and properly managed.

Table 1. "How useful do you think are the following features of the SPARK environment?"



	Easy setup (Phone and PC software)	Having a remote console	Edit project files on PC and execute them on the phone	Package files and create distributable SIS files	Test programs parallel on different phones	Already included extension modules
Highly useful	8	14	22	11	5	13
Useful	18	8	5	8	7	7
Somewhat useful	1	5	0	3	2	4
Not useful	0	0	0	0	0	1
N/A	0	0	0	5	13	2

Ensymble [8] is a development project offering a set of command line tools for packaging PyS60 files into standalone SIS applications. SPARK uses the Ensymble tool set and extends it with additional features to reduce the amount of know-how required by the developer. In addition to providing a GUI to Ensymble, SPARK adds the following features: Choosing modules that have the appropriate capabilities and have been signed using the right certificate; choosing a correct application UID; icon creation; code parser for recommending external modules to include; and the auto-update mechanism (cf section 3.1).

5 Use Case: Use in Teaching

We used the SPARK rapid prototyping environment in the fall of 2008 as part of a lecture on distributed systems. The course had 73 enrolled graduate students, all from computer science. The topic "mobile phone programming" was not covered in the lecture, and students were not required to have had prior courses in mobile phone programming. Instead, the idea was to have students explore some of the concepts of distributed systems in a hands-on manner by programming a few applications on a mobile phone. Students had 3 weeks to design and implement a project using PyS60. Students were given access to the SPARK environment, but were free to use it or not (and instead rely on the standard PyS60 tools). Due to the limited time available for the two exercises, students were given a 30 minute introduction to PyS60/SPARK. They were then asked to form groups of two to work on the exercises. We had 50 Nokia N95/8g at our disposal[5], so each group was given one or two devices. Students were free to take the devices home during the exercise period. Feedback was gained in multiple ways: After the end of the second exercise, students were asked to present their applications to their peers (and to us). We also encouraged students to approach us with questions during the exercises, and asked them to fill out an online questionnaire with 26 questions about PyS60, and the SPARK rapid prototyping environment at the end of the course. We received 30 answers to the anonymous questionnaire (36% return rate). Last but not least, all SPARK environments by default logged all system events and user interactions with the software into a simple text file. Participants were informed about this logging, had the chance to review this

[5] 30 devices were graciously sponsored by Nokia.

data, and were asked to provide us with this file on a voluntary basis. 25 students sent us these log files. Here we can only briefly summarize the results. The full questionnaire and the results can be accessed from <removed for blind review>. All respondents (n=30) ended up using SPARK and only 2 students also used alternative methods (non-SPARK based) to develop their PyS60 application.

Entry barriers: Users were asked how easy it was for them to get started with PyS60 development using SPARK. The majority of them found it very easy (51.9%) and easy (44.4%). We also asked students for installation problems with the SPARK environment. 15 students answered this optional question, with 13 stating they had no problems and 2 stating that they had Bluetooth issues on Windows Vista and Linux.

SPARK features: Table 1 presents how useful students rated the various features of SPARK for their project. Support for automated SIS package generation and for concurrent development on different device types were rated as being not that important. Since these features address problems occurring in real-world deployments, this comes as no surprise. Asked to tell us what kind of features they missed, most students stated that they lacked a real PyS60 debugger (77.8%).

General Feedback: The majority of students found the SPARK environment highly useful (66.7%) or useful (25.9%) for realizing their project. All students indicated that they would use the software again and that they would recommend it to colleagues. Answers to the open question about what they liked most about the software included: “Easy to use; good looking; worked instantly”, “It makes the development very easy and fast.”, “The time you need to deploy and test the code is very short.”, “Easy, simple, fast, reliable, free to use”. Things that students disliked about SPARK were: “Bluetooth problems” and “Missing built-in help”.

6 Conclusion

There is a growing gap between the emerging mobile phone platforms with their sophisticated capabilities, and the expertise among non-expert developers to control them. This gap cuts off an important source of creativity, as many might be inspired to create novel types of mobile applications using the phones’ increasing power, but only few have the time and energy to dig into the intricacies of low-level mobile device programming. In this paper we presented a comprehensive overview of current rapid prototyping tools for mobile phones and identified PyS60 as an attractive choice that combines ease of use with flexibility. However, PyS60 development environments still pose considerable barriers for the rapid prototyping of applications, especially for novel users and when targeting large-scale, real-world deployments. We identified these barriers and presented the SPARK tool, which targets non-expert users and supports fast development cycles. Due to its low entry barrier, it is highly suitable for use in an educational environment. We also presented the results of a large case study in which SPARK has been used – a graduate course on distributed systems, in which 70+ students used SPARK to develop mobile applications, with the large majority reporting the tool to be highly useful. SPARK is available as a free download at http://people.inf.ethz.ch/adelmanr/spark.

References

1. Adelmann, R.: Mobile Phone Based Interaction with Everyday Products - On the Go. In: Proc. of NGMAST07. IEEE Computer Society, 63-69 (2007)
2. LaMarca, A., et al.: Place lab: Device positioning using radio beacons in the wild. In: Proc. of PERVASIVE 2005, LNCS 3468, 116–133 (2005)
3. Nicolai, T., Yoneki, E., Behrens, N., Kenn., H.: Exploring Social Context with the Wireless Rope. In: Proc. of MONET'06, Montpellier, France, 874-883 (2006)
4. Python for S60 Open Source Project, http://sourceforge.net/projects/pys60
5. Symbian Ltd. Symbian Signed User Guide, www.symbiansigned.com/app/page
6. J2ME Java 2 Micro Edition, http://java.sun.com/javame/index.jsp
7. Harrison, R.: Symbian OS C++ for Mobile Phones, Wiley (2003)
8. The ensymble developer utilities for symbian os, http://code.google.com/ p/ensymble/
9. Python utility tools for pys60, http://people.csail.mit.edu/kapu/symbian/python.html
10. Laurila, J., Tuulos, V., MacLaverty, R.: Scripting Environment for Pervasive Application Exploration on Mobile Phones. In: Proc. of PERVASIVE 2006, LNCS, Dublin (2006)
11. Metro group MEA application, www.future-store.org/fsi-internet/html/de/7803/index.html
12. BlueCove library for Bluetooth (JSR-82) implementation), www.bluecove.org/
13. Holleis, P., Schmidt, A.: MakeIt: Integrate User Interaction Times in the Design Process of Mobile Applications. In: Proc. of PERVASIVE 2008, LNCS Sydney, 56-74 (2008)
14. Joki, A., Burke, J. A. and Estrin, D.: Campaignr: A Framework for Participatory Data Collection on Mobile Phones. Center for Embedded Network Sensing. TR 770 (2007)
15. Raento, M., Oulasvirta, A., Petit, R., Toivonen, H.: Contextphone: a prototyping platform for context-aware mobile applications. Pervasive Computing, IEEE 4, 51-59 (2005)
16. Forstner, B., et al.: Supporting Rapid Application Development on Symbian Platform. In: Proc. of EUROCON 2005, IEEE Computer Society, Belgrade, Serbia, 72-75 (2005)
17. Long, S., et al.: Rapid prototyping of mobile context-aware applications: the Cyberguide case study. In: Proc. of MobiCom '96. ACM, 97-107 (1996)
18. Adelmann, R., et al.: Toolkit for Bar Code Recognition and Resolving on Camera Phones – Jump Starting the Internet of Things. In: Proc. MEIS'06, 366-373 (2006)
19. Huebscher, M., et al.: Issues in Developing Ubicomp Applications on Symbian Phones. In: Proc. FUMCA'06. IEEE Computer Society, 51-56 (2006)
20. Card, S.K., Moran, T.P., Newell, A.: The Keystroke-Level Model for User Performance Time with Interactive Systems. In: Comm. of the ACM, 23(7), 396–410 (1980)
21. Mohan, P., Padmanabhan, V.N., Ramjee, R.: Nericell: Rich monitoring of road and traffic conditions using mobile smartphones. In: Proc. of SenSys '08, ACM, 323–336 (2008)
22. Gartner Report on Smartphone Sales, www.gartner.com/it/page.jsp?id=827912
23. J2ME Polish, www.j2mepolish.org/cms/
24. Forum Nokia Python Resources, http://wiki.forum.nokia.com/index.php/Category:Python
25. Scheible, J., Tuulos, V.: Mobile Python: Rapid Prototyping of Applications on the Mobile Platform. Wiley (2007)
26. Objective C, http://developer.apple.com/documentation/Cocoa/Conceptual/ObjectiveC/
27. Lua for S60 devices, http://luaforge.net/projects/luas60/
28. Ruby for S60 devices, http://ruby-symbian.rubyforge.org/
29. FlashLite, www.adobe.com/products/flashlite/
30. Hecl – The Mobile Scripting Language, www.hecl.org/
31. Froehlich, J., et al.: MyExperience: a system for in situ tracing and capturing of user feedback on mobile phones. In: Proc.of MobiSys '07, ACM, 57-70 (2007)
32. Yang, L., & James, A. L.: Activity-based prototyping of ubicomp applications for long-lived, everyday human activities. In: Proc. of SIGCHI, 1303-1312 (2008)
33. Fielding, R. T.: Architectural Styles and the Design of Network-Based Software Architectures. Doctoral Thesis. UMI: AAI9980887, University of California, Irvine (2000)