A CASE Tool for Java Mobile Computing Applications

Ioannis T. Christou, Athens Information Technology, Greece
Sofoklis Efremidis, Athens Information Technology, Greece
Aikaterini Roukounaki, Athens Information Technology, Greece

ABSTRACT

The market for applications developed for mobile devices is growing as the hardware capabilities increase while costs drop. At the same time, the inability to write code once becomes a noticeable problem resulting in the time consuming task of porting an application to a variety of mobile devices, which is true for languages like Java that have been designed to be portable across a range of hardware platforms. As a result, porting a Java application written for one device to another is often a tedious and time-consuming task for developers. This paper presents an intelligent CASE tool that assists the porting of Java mobile applications by automatically generating Java code fragments for the target SDK. SeqFinder automatically generates all minimal method invocation sequences that lead to an object of a specific type, thus relieving the programmer of manually searching the manufacturer-provided SDK Java archives. However, this tool is not applicable only to the Java ME platform and has been used as a fast type-browser for J2SE/J2EE applications.

Keywords: Automatic Program Generation, CASE, CASE Tools, Java, Mobile Technologies

INTRODUCTION

Mobile Computing has witnessed significant growth in the past two decades, since its initial conceptualization. While the focus of attention has often been the mobile laptop user with roaming WIFI connections (Satyanarayanan, 1996; Bellavista et al., 2001; Pittoura & Samaras, 2001), lately there has been significant focus on developing applications for constrained mobile devices such as smart-phones and Personal Digital Assistants (PDAs) (Sohn et al., 2006; Tomek & Shakshuki, 2009; Sugumaran et al., 2009; Rawassizadeh, 2009).

Application development has always been a challenging task that becomes even more difficult for applications targeting mobile devices where porting issues often prevail. The inability to write once, run everywhere can be attributed to two main reasons:

1. Hardware diversity, i.e., differences in hardware characteristics among mobile devices, like screen size, color depth, memory size, support for peripheral devices like cameras, Bluetooth connectivity, etc.

DOI: 10.4018/jmcmc.2010040103
2. Software diversity, which refers to
   a. Differences in the underlying operating
      system (Symbian, PalmOS, etc.)
   b. Differences in the supported standards,
      i.e., supported versions of the Con-
      nected (Limited) Device Configura-
      tion (“Connected Limited Device
      Configuration (CLDC)”, 2009) and
      Mobile Information Device Profile
      (“Mobile Information Device Profile
      (MIDP),” 2009; Ortiz & Giguere,
      2001) of the Java 2 Mobile Edition
      (“Java ME – The Most Ubiquitous
      Platform for Mobile Devices”, 2009)
   c. Differences in the implementation,
      i.e., vendor specific implementations
      of MIDP or CLDC APIs, or even
      vendor/device/series specific libraries
      and APIs extending or complementing
      those specified in the standard CLDC
      and MIDP, for example codecs, graph-
      ics support APIs, etc.

   It turns out that this diversity is rather
difficult to mask off from application
developers. Contrary to the desktop and server
world, where Java Virtual Machine are almost
universally based on SUN’s Java SDK (j2sdk)
regardless of Operating System and computer
architecture of the machine running the JVM,
the Java Kilobyte Virtual Machines (KVM)
installed in the mobile phones of today are the
manufacturer’s implementation of the KVM
specification. This practically means that each
mobile device runs a different implementation
of the same specification than any other mobile
device of another manufacturer. Implementation
bugs, combined with the fact that the specifica-
tion does not always specify completely what
a functionality or Application Programming
Interface (API) is supposed to do, have lead to
an application development and maintenance
nightmare for third-part developers of mobile
applications. It is an unfortunate fact therefore,
that the promise of Java (Gosling, Joy, Steele,
& Bracha, 2000) as a “write once, run every-
where” language proved to be overly optimistic
in the mobile phone market-place. This is by
no means a criticism on the language itself, as
Java is probably the best-suited language for
writing applications for mobile phones and
mobile information devices in general, but
rather a realization that the extreme variation
in those devices’ characteristics necessitated
different versions of the language specifically
tailored to these devices. Undoubtedly, the area
with the most variations in the offered APIs is
the Graphical User Interface, but as different
devices support different features and offer
the corresponding APIs for developers to take
advantage of, there is a wide range of capabili-
ties of different devices and, correspondingly,
of the APIs supporting them.

   Even standardized APIs such as the con-
nectivity API that allows access to the Internet
and servers and services available over HTTP or
socket protocols are often poorly implemented
by the device manufacturers and cause serious
porting problems to application developers. To
further complicate things, many (early) phone
models that claimed to support the MIDP speci-
ification had serious bugs in the built-in Kilobyte
Virtual Machine (KVM) especially regarding
synchronization issues, thread management, and
memory management. These defects combined,
makes writing complex applications for such
devices at times a daunting task.

   Porting of existing applications from one
device type to another (of the same or different
vendor) is an equally complicated process. The
particularities of the APIs as well as the KVM
and MIDP implementations of the concerned
device types have to be taken into account,
making porting a time consuming and tedious
process. Typically, application developers
and/or application porters need to try several
alternative sequences of object constructions
and method invocations for a particular task at
hand. For example, the code sequences for the
construction of an InputStream object for read-
ing data from a source differ depending on the
supporting MIDP implementation libraries. Any
Related Content

Adoption of M-Commerce Devices by Consumers
[www.irma-international.org/chapter/ado.../17049/](http://www.irma-international.org/chapter/ado.../17049/)

LTE-A Implementation Scenarios: RF Planning Comparison
[www.irma-international.org/article/lte-implementation-scenarios/63049/](http://www.irma-international.org/article/lte-implementation-scenarios/63049/)

Service Delivery Platforms in Mobile Convergence
[www.irma-international.org/chapter/service-delivery-platforms-mobile-convergence/17188/](http://www.irma-international.org/chapter/service-delivery-platforms-mobile-convergence/17188/)

Semantic Handover among Distributed Coverage Zones for an Ambient Continuous Service Session
[www.irma-international.org/article/semantic-handover-among-distributed-coverage/76308/](http://www.irma-international.org/article/semantic-handover-among-distributed-coverage/76308/)

A Cross-Layer Model for Video Multicast Based TCP-Adaptive FEC over Heterogeneous Networks
[www.irma-international.org/article/cross-layer-model-video-multicast/4063/](http://www.irma-international.org/article/cross-layer-model-video-multicast/4063/)