組込み仮想化を用いたソフトウェアテスト方法

2011SE207 追良瀬 利也 2011SE292 山田 俊之介

指導教員:青山 幹雄

1 はじめに

スマートデバイス向けのソフトウェア開発では，機種やOS のバージョンの違いにより互換性の問題が発生する．開発現場では最終的な実機テストを行うために必要な機種とOS の組み合わせの実機を用意する必要がある．OS のバージョンが異なる場合では，同一の機種でもそれぞれの OS で同様のテストを実行しなければならないのでテスト工数が増加する[4].

2 研究課題

同一機種のソフトウェアテストで異なる版数のOS の互換性を確認するには，複数の実機にそれぞれ異なる版数のOSを起動させ，ソフトウェアを実行する．そのため，複数の実機を用意しなければならない．

そこで，本研究の課題を以下とする．

(1) アーキテクチャの提案

同一機器上で異なる版数のOSでのソフトウェアテストを可能とするアーキテクチャを実現する．

(2) 提案するアーキテクチャの妥当性の確認

プロトタイプを用いて実際のソフトウェアテストを行うことで，アーキテクチャの妥当性の確認を行う．

3 関連技術

組込み仮想化とは，組込み機器にハイパーバイザを導入し，複数のOSを単一のハードウェア上で稼働させる手法である．組込み仮想化に利用されるハイパーバイザは，他分野で用いられるハイパーバイザとは異なり以下の制約がある[1].

(1) 効率性

組込み機器のコンピュータリソースには制限があるので，組込みの仮想化に用いられるハイパーバイザは小規模でなければならなく，特にメモリの使用が効率的である．

(2) セキュリティ

コードサイズが小規模であるので，安全で信頼できるプラットフォームである．

(3) リアルタイム対応

コンピュータリソースの消費が少ない設計がされているので，仮想化する際のオーバヘッドが仮想化以前の組込み機器に OS を搭載した場合とそれほど違いはない．

(4) 分離性

ハイパーバイザ上のゲスト OS は独立した環境に共存できる．また，ゲスト OS に割り当てられたコンピュータの資源は OS 同士で競合しない．

4 アプローチ

本研究では，組込み仮想化技術を用いて版数が異なる複数のOSを単一の機器に集約することで開発の効率化を図る．特に，実機でのテスト段階に着目し，工数を削減する．

現在，国内での主なスマートフォン用OSはAndroid，iOS，Windows Phoneがあるが，ハイパーバイザを実機に組込むので，OSのソースコードが公開されており，修正可能なオープンソースであるAndroidを本研究の対象のOSとする．

5 提案方法

5.1 提案方法のアーキテクチャ

単一の機器に仮想化環境を実装し複数のプラットフォームを構築することで，複数の端末を用意することなく異なるプラットフォームでのアプリケーションのテストを可能にする．アーキテクチャと構成要素を以下に示す．

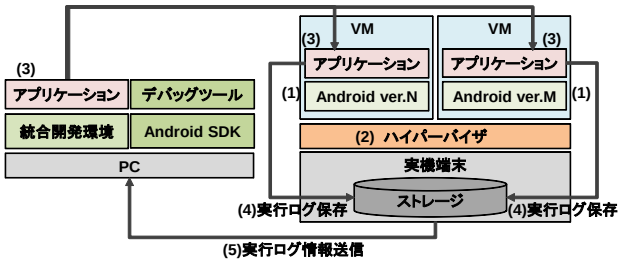

図 1 提案方法のアーキテクチャ

(1) プラットフォーム

異なる OS のバージョンで動作したアプリケーションの実行ログを収集するので，異なる版数の Android を使用する．

(2) ハイパーバイザ

組込み機器用に開発されたハイパーバイザである．実機端末の CPU アーキテクチャに対応している必要がある．

(3) アプリケーション

テストの対象となるアプリケーションである．単一の機器で異なる版数の OS での実行を確認できる．

(4) 実行ログ保存

各バージョンの OS で実行されたアプリケーションの実行ログを一時ストレージに格納する．

(5) 実行ログ情報送信

ストレージに一時格納した実行ログの情報を PC へ送信する．本研究の実行ログの解析には Android SDK に付属しているTraceview[3]を使用することを前提とする．

実行ログを解析した結果を基にソフトウェアのデバッグが可能．

5.2 提案アーキテクチャの振る舞い

仮想化環境の振る舞いを図 2 に示す.

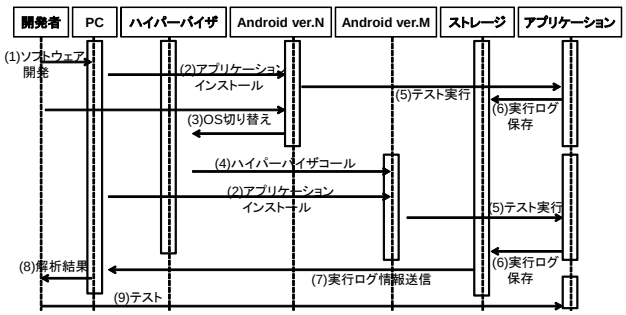

図 2 仮想化環境のシーケンス図

(1) アプリケーション開発

アプリケーションを PC 上で開発する. エミュレータを用いたテストは行わない.

(2) アプリケーションインストール

単一の機器上で異なる版数の OS にネットワークを経由してダウンロード可能とする.

(3) OS 切り替え

Android ver.N でアプリケーションを実行し, 実行ログをストレージに格納した後, 異なるプラットフォームでテストをするために OS を Android ver.M に切り替える.

(4) ハイパーバイザコール

ハイパーバイザからの命令により OS を切り替える. これにより, ハードウェアを制御するのは Android ver.N から Android ver.M になり, 最初にテストした環境と異なる環境でのテストが可能になる.

(5) テスト実行

各バージョンの OS でアプリケーションを実行させる.

(6) 実行ログ格納

各バージョンの OS で実行したアプリケーションの実行ログをストレージに格納する.

(7) 実行ログ情報送信

ツールを用いて解析するために PC へ実行ログの情報を送信する.

(8) 解析結果

PC 上で解析した結果を表示する. 異なる OS で実行されたアプリケーションの実行ログを比較する.

(9) テスト実行

解析結果を基にテストを実行する. 互換性を保証するために異なるバージョンで実行しても差が出ないアプリケーションに近づける.

5.3 提案方法を用いた開発プロセス

提案方法を用いたアプリケーション開発の全体のプロセスを以下に示す.

(1) アプリケーション開発

アプリケーションの開発者は要求定義や設計が行われたアプリケーションを PC 上で開発する. 実機テストまでの開発は PC 上で行う.

(2) 各 OS にアプリケーションをインストール

PC 上で開発したアプリケーションを, 実機テストを行う端末の各バージョンの OS に, ADB コマンドによりネットワーク経由でインストールする.

(3) 異なるプラットフォームでのテスト

最初にテストする OS である Android ver. N を起動する. 作成したテストケースに基づき, テストを実行する. 次に Android ver.M に切り替えテストを実行する.

(4) ログ情報の送信

ストレージに格納されている実行ログの情報を PC に送信する.

(5) ログ解析

ストレージに格納されているログ情報を PC 上でツールを用いて解析する.

(6) ログ解析の結果の正誤

(5)で解析/比較した結果が正しければ本プロセスを終了し, 正しくなければアプリケーションを修正する. 修正したアプリケーションを再度端末の各 OS にインストールし, 解析結果が正しくなるまで(3)-(6)を繰り返す.

本研究では, ログ解析ツールに Traceview を使用することが前提なので, 実行ログ送信の出力にはトレースファイルが出力される. ログ解析ツールに Traceview を使用した提案方法のプロセスの概要を図 3 に示す.

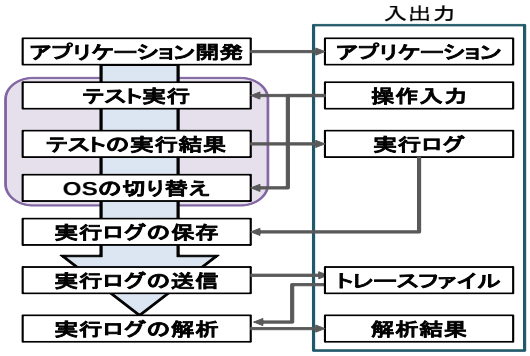

図 3 提案方法を用いた開発プロセス

6 プロトタイプの実装

6.1 プロトタイプの目的

プロトタイプを用いて, 単一の機器上の異なる OS でアプリケーションを実行し, 仮想化環境上での提案するテスト方法の妥当性を確認する.

PC 上に仮想化環境を実装し, 異なるバージョンの Android を実装することで異なるプラットフォームでの実機テストの妥当性を評価する. PC 上の仮想化環境は, 組込み仮想化に近づけるためにハイパーバイザ型の仮想化方式を採用する. また, 仮想化環境上の Android は, x86 アーキテクチャに対応したものを用いる.

6.2 プロトタイプの機能

アプリケーションのテストは, 画面遷移のテストを想定する. 実際に, Eclipse を用いて Android アプリケーションを開発し, 仮想化環境を実装した PC にインストールすることで実機でのテストを再現する. 図 4 にプロトタイプのユースケースを示す.

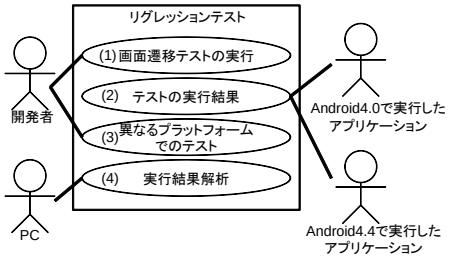

図 4 プロトタイプのユースケース

6.3 プロトタイプの実装環境

プロトタイプの仕様を表 1 と表 2 に示す.

表 1 仮想化環境の仕様

CPU	Intel(R) Core(TM) i5 (仮想化支援機能対応)
ハイパーバイザ	Xen 4.2
仮想化方式	完全仮想化
Domain-0	CentOS 6.5
Domain-U	Android4.0 (メモリ 2G)
Domain-V	Android4.4 (メモリ 2G)

表 2 開発環境の仕様

OS	**Windows**
SDK	**Android SDK**
統合開発環境	**Eclipse**
ログ解析ツール	**Traceview**

6.4 プロトタイプの実装

実装したプロトタイプのアーキテクチャを図 5 に示す.

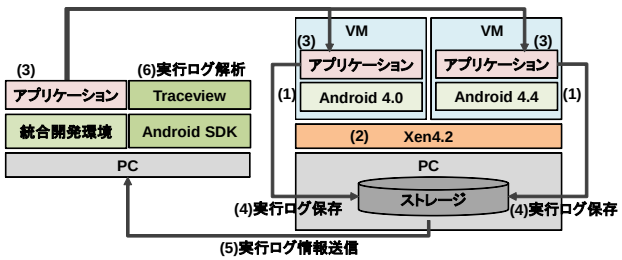

図 5 プロトタイプの実装

(1) プラットフォーム

版数が異なる OS で実行したアプリケーションの実行ログを収集するために, Android4.0 と Android4.4 の x86 アーキテクチャに対応したものを使用する. これにより, 異なる版数の OS のテストを単一の機器で可能であることを確認する.

(2) Xen4.2

一般的なハイパーバイザ型である Xen を使用する. プロトタイプでは, x86 アーキテクチャに対応した Android をプラットフォームとして用いるので, CPU の仮想化支援機能を使用した完全仮想化を採用する. Xen のネットワークモードであるブリッジモードにより外部との通信をする [2][5]. この通信により, 開発用 PC で作成した Android アプリケーションをネットワーク経由で, PC に実装した仮想化環境にインストールする.

(3) アプリケーション

ボタンを押すことにより画面が切り替わるアプリケーションである. プロトタイプは PC で実装するので, 通話機能やカメラ機能を使用するアプリケーションは使用しない.

(4) 実行ログの保存

Android SDK に付属する Traceview を使用し, 実行ログをトレースファイルとして SD カードに保存する.

6.5 テスト対象のアプリケーション

プロトタイプのテスト対象のアプリケーションの概要を以下に示す.

(1) 4 つの画面を遷移する.

(2) それぞれの画面には他の画面へ遷移する 3 つのボタンが配置されている.

(3) それぞれの画面の名前は画面 1, 画面 2, 画面 3, 画面 4 である.

6.6 プロトタイプの実行結果

テスト対象のアプリケーションを Xen 上の Android4.0 と Android4.4 にインストールし, 画面遷移のテストを実行した. そして, ボタンを押すことで実行されるメソッドの実行ログを収集し解析した.

アプリケーションを 10 回実行し, 解析結果を基に異なる版数の OS で実行したアプリケーションを比較する.

解析結果を基に比較するメソッドを決定した. アプリケーションを実行した際の実行時間を比較するために表 8-3 の各メソッドは Android4.0 と Android4.4 の共通のメソッドを選択した(表 3). 各メソッド実行の実行時間を表 4 と表 5 に示す.

表 3 CPU 処理時間の長いメソッド

A	AssetManager.applyStyle(IIII[I[I[I)Z
B	AssetManager.openXmlAssetNative (ILjava/lang/String;)I
C	ViewGroup$MarginLayoutParams.<init> (Landroid/content/Context;Landroid/util/AttributeSet;)V
D	View.<init> (Landroid/content/Context;Landroid/util/AttributeSet;I)V
E	TextView.<init> (Landroid/content/Context;Landroid/utilAttributeSet;I)V
F	ContextImpl.getApplicationInfo()Landroid/content/pm/ApplicationInfo;
G	ContextWrapper.getApplicationInfo()Landroid/conetnt/pm/ApplicationInfo;
H	Paint.native_setTextLocale (ILjava/lang/String;)V
I	Resources$Theme.obtainStyledAttributes (Landroid/util/AttributeSet;[III)Landroid/content.res/TypedAfrray;
J	View.setFlags (II)Z

表 4 Android4.0 の実行結果

回目 / メソッド	1	2	3	4	5	6	7	8	9	10	平均	標準偏差
A	0.58	0.56	0.63	0.60	0.73	0.59	0.52	0.46	0.57	0.56	0.57	0.066
B	0.41	0.42	0.46	0.31	0.43	0.42	0.35	0.29	0.38	0.28	0.38	0.061
C	0.26	0.27	0.31	0.28	0.29	0.28	0.32	0.22	0.24	0.21	0.24	0.035
D	0.41	0.39	0.45	0.42	0.43	0.43	0.43	0.32	0.40	0.34	0.40	0.038
E	0.41	0.46	0.44	0.49	0.53	0.41	0.39	0.39	0.43	0.38	0.43	0.047
F	0.32	0.32	0.36	0.49	0.36	0.32	0.37	0.29	0.35	0.31	0.35	0.055
G	0.30	0.34	0.33	0.31	0.33	0.32	0.32	0.27	0.29	0.27	0.29	0.024
H	0.30	0.28	0.31	0.30	0.43	0.36	0.26	0.22	0.29	0.28	0.29	0.053
I	0.27	0.31	0.29	0.27	0.30	0.27	0.28	0.29	0.24	0.21	0.24	0.027
J	0.26	0.26	0.25	0.26	0.29	0.26	0.25	0.24	0.26	0.23	0.26	0.015

表 5 Android4.4 の実行結果

回目 / メソッド	1	2	3	4	5	6	7	8	9	10	平均	標準偏差
A	0.59	0.64	0.56	0.54	0.41	0.42	0.63	0.59	0.58	0.52	0.55	0.074
B	0.43	0.47	0.37	0.40	0.25	0.31	0.45	0.35	0.34	0.41	0.40	0.064
C	0.41	0.31	0.25	0.22	0.21	0.21	0.32	0.25	0.22	0.30	0.27	0.061
D	0.37	0.43	0.36	0.33	0.36	0.32	0.48	0.39	0.33	0.37	0.37	0.047
E	0.37	0.44	0.39	0.42	0.38	0.39	0.38	0.46	0.37	0.40	0.40	0.030
F	0.32	0.37	0.29	0.29	0.28	0.28	0.40	0.32	0.29	0.29	0.31	0.040
G	0.32	0.34	0.31	0.26	0.28	0.27	0.36	0.31	0.29	0.31	0.30	0.030
H	0.29	0.31	0.25	0.32	0.25	0.22	0.28	0.38	0.22	0.32	0.28	0.047
I	0.25	0.30	0.23	0.22	0.20	0.21	0.30	0.25	0.23	0.22	0.24	0.033
J	0.25	0.27	0.22	0.22	0.21	0.23	0.30	0.25	0.23	0.24	0.24	0.026

各メソッドの標準偏差から各数値の統計的な違いはない．また，テストによって意図した画面遷移が行われた．この二つの観点から，仮想化環境上の Android4.0 と Android4.4 共にアプリケーションの実行は正常に行われたことを確認した．よって，ソフトウェアテストに利用できるデータの収集が可能であるといえる．

収集したデータを基に，実際に異なる版数で実行したアプリケーションのログ解析結果を比較する．比較結果を図 6 に示す．

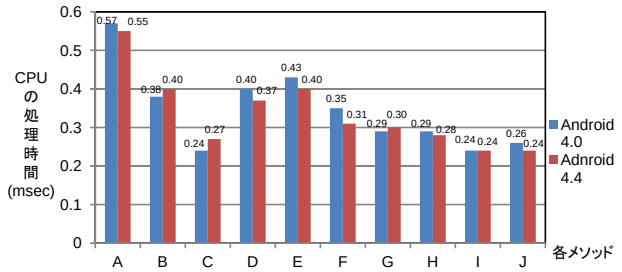

図 6 処理時間の比較

図 6 より，各メソッドの実行時間の差は±0.04msec 以内であり，統計的に有意の差は見られない．すなわち，異なる版数の OS で今回実行したアプリケーションのパフォーマンスの変化は見られないことを確認した．

プロトタイプを用いてソフトウェアテストの一連の流れを実行することで，同一プラットフォーム上で異なる版数の OS でのアプリケーションの互換性に関するテストが可能であることを示した．

6.7 プロトタイプの評価

PC でのプロトタイプを用いて，提案方法で示した仮想化環境のアーキテクチャを実現した．プロトタイプによって，異なる版数の OS で実行したアプリケーションの実行ログの解析や，開発の一連の流れを実行することによって提案方法の妥当性を確認した．

本研究のプロトタイプでは組込み機器を用いずに PC でプロトタイプを実装した．端末でのタッチ操作ではなく PC でのマウス操作になること，ハイパーバイザの違い，CPU 性能，メモリ容量，仮想化環境上の OS に Android の x86 アーキテクチャ対応のものを用いるといった実際の組込み機器での実装と違いがある．しかし，異なる版数の OS を起動させることで，同一の機器上で異なるプラットフォームでのテストが可能であることを示した．

Xen には，ARM アーキテクチャ対応のものがあることや，Xen の機能により CPU のコア数やメモリ容量を実機端末の性能に近づけるよう割り当てたことから，PC でのプロトタイプは有効であったと考えられる．

7 考察

7.1 プロトタイプに関する考察

仮想化技術を用いて PC 上に版数が異なる 2 つの OS を実装することで，異なる版数の OS でのアプリケーションテストが可能であると確認することができた．本研究では，アプリケーションのテストは画面遷移にのみに着目したが，センサやカメラなどのデバイスを使用したアプリケーションのテストに提案方法を適用させた場合の評価も必要であると考える．

また，実行結果の比較としてアプリケーションの各メソッドの処理時間を解析したが，スマートデバイスのアプリケーションテストとして，ユーザビリティテストや異なるシステムやソフトウェアとの相互作用に関するテストでの妥当性の確認が必要であると考えられる．

7.2 提案方法に関する考察

本研究では，組込み仮想化環境を用いたスマートデバイスのテスト方法を提案した．従来のソフトウェアテストでは，OS のバージョン毎に複数の端末を用意する必要があった．組込み仮想化技術を用いて異なる版数の OS を同時に起動させることで，単一の実機で異なるプラットフォームでのテストが可能になる．ソフトウェアテストを効率化させることで，OS のアップデートのサイクルが早い場合でも，互換性の高いソフトウェア開発が可能になると考えられる．

仮想化環境上に複数の OS を起動するには，実機端末のメモリ性能が重要になる．今後もスマートデバイスのメモリ性能の向上が予想されるので，仮想化環境上でより多くの OS が実装可能であると考えられる．よって，同時にテストできる環境が複数構築できるので，必要とする実機端末の削減が可能である．

8 今後の課題

(1) 実機端末での実装
(2) 実行画面の比較
(3) 挙動の影響の確認

9 まとめ

本研究では，組込み仮想化技術を用いてスマートデバイス上に異なる OS を実装し，単一の機器で異なるプラットフォームでの実機テスト方法を提案した．

本研究では，単一の機器で異なるプラットフォームでの実機テスト方法を可能にするアーキテクチャのうち，仮想化環境を Xen と x86 アーキテクチャに対応した Android でプロトタイプを実装した．開発したアプリケーションを仮想化環境上の Android で実行し実行ログを収集することで提案するテスト方法の妥当性の確認と評価を行った．

10 参考文献

[1] A. Vasudevan et al., Trustworthy Execution on Mobile Devices, Springer Briefs in Computer Science, pp37-48, 2013.
[2] D. Chisnall, Definitive Guide to the Xen Hypervisor: 1st Edition, Prentice Hall, 2007.
[3] 木田学他，Android アプリを作る 139 の鉄則，技術評論社，2014.
[4] 斎藤 栄太郎，中山 秀夫，モバイルアプリ開発の新潮流，日経SYSTEMS，2013年8月号，pp. 34-37.
[5] 平 初 他，仮想化技術完全攻略ガイド，インプレスコミュニケーションズ，2006.