PATHPARTNER

CUSTOMIZING ANDROID UI

To functionalise, optimize & differentiate

Customizing android from user interface perspective is a hot requirement from device makers and overall solution providers. Android is set to be on an exponential growth both in smart-phone and non-smartphone market. In this report we discuss the need for customization and challenges. Pathpartner's mediaphone solution sports a completely customized UI with custom back-end services. Main emphasis is given on portability to future android versions, by limiting most of the changes to published APIs in the SDK.

Alexy Mathew Joseph, Ravi Pandit & Vinay M K

9/22/2010

www.pathpartnertech.com

CUSTOMIZING ANDROID UI

To functionalise, optimize & differentiate

A of Android

Android has taken the smart-phone market by storm since its official launch on Sep 2008 in North America by HTC's Dream (T-Mobile G1). Android's market share in smart-phone market in North America has been rising steadily as estimated by Quantcast, based on mobile phone web traffic. Android's popularity is not just limited to mobile phone market, instead it has caught on like a wild fire in non-smart-phone markets like tablets, TVs, Kiosks, etc., Major OEMs like Samsung and LG have announced Android based tablets, while Sony is working with Google in bringing Android based Google-TV in 2011. Networking equipments company Cisco has announced Android based video conferencing device early next year. Android is going to be relevant both in smart-phone as well as non-handset (CE) markets.

What makes Android tick?

There are various factors contributing to Android's proliferation making it a preferred choice among mobile OEMs and CE device makers. Primary among them are the completeness of the phone stack and Apache licensing of user space modules. Bulk of the essential phone stack modules have been commercially bought and integrated into Android. These commercial modules are optimal and feature rich to a fair degree and combined with their apache licensing terms, are available for free to device makers, with no strings attached open for modifications & innovation.

Unlike LiMo phones which lacks completeness in eco-system (Phone stack, app-store, XMPP), Android brings in LiMo + phone-stack + App-store. Incidentally it also brings in Apache licensed modules allowing device makers to include their customizations without having to give it away to open-source. Modularity of android's user stack elements accelerates its customization to suit device makers target requirements. Even though Android is designed for smart-phone devices, its apache licensed user space, growing community support, rich feature set and growing R&D investment from OHA members makes it a BoM friendly choice for low-cost consumer electronics devices. For CE designers who are used to embedded Linux layered with GTK, Android offers additional advantages.

Android is a BoM friendly choice for low-cost consumer electronics device makers. Modularity of user stack elements makes android customization easy and

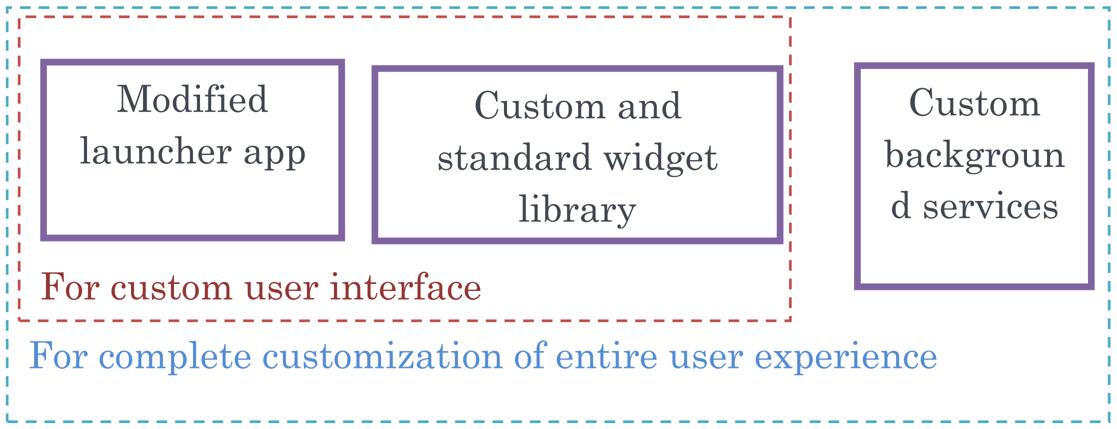
Why Android is customized?

A device maker, by choosing Android, would still have the business need to differentiate from other android based devices by customizing atleast the basic user interface and where ever viable the entire user experience. This trend is evident from the early adopters of Android in smart-phone market eg., HTC's Sense UI, Motorola's MOTOBLUR, Sony's Ux & Samsung's TouchWiz. Some of these customizations are beyond simple skin changes, instead they require dedicated services running in the background to feed these custom UI widgets with data and context. Directly customizing "built in" UI modules or accessing not-yet released private-APIs in Android, brings with it a set of compatibility, cost and delayed-updation issues. Android is still in its early days of evolution and new-features, optimizations and functionalities are getting added almost every day. Major releases are planned every six months, with improvements in performance as high as 4-5 times over the previous release. Customization based on private APIs not published in Android SDK would mean additional effort and cost for upgradation with new OS releases. With many low-cost Android smart-phones planned by major OEMs like Huwaei (IDEOS), ZTE (Racer), Acer (neotouch), Gigabyte (Codfish) etc., this problem is bound to reverberate across the industry. Usage of android in non-smartphone devices like TVs, Mediaphones, Tablets, e-reader would need more of these customizations not just to differentiate but also to meet basic functionality requirements on the target device. Default android homescreen may not be suitable for a Set-op box device, instead a program guide (EPG) may be more relevant.

Usage of android in non smartphone devices like TVs, Mediaphones, Tablets, eReader would need more of these customizations not just to differentiate but also to meet functionality requirements on the target device

Customizing Challenges in Android

For both smart-phone and non-smart-phone devices it is highly desirable to implement custom UI designs without using Android's private APIs as much as possible. Home screen customization in Android is achieved by replacing the standard launcher.java app with a custom made application. Entire user experience customization would usually need modifications and development effort at all the three components; launcher app, widget library & relevant background services.


Modified launcher app
Custom and standard widget library
Custom backgroun d services
For custom user interface
For complete customization of entire user experience

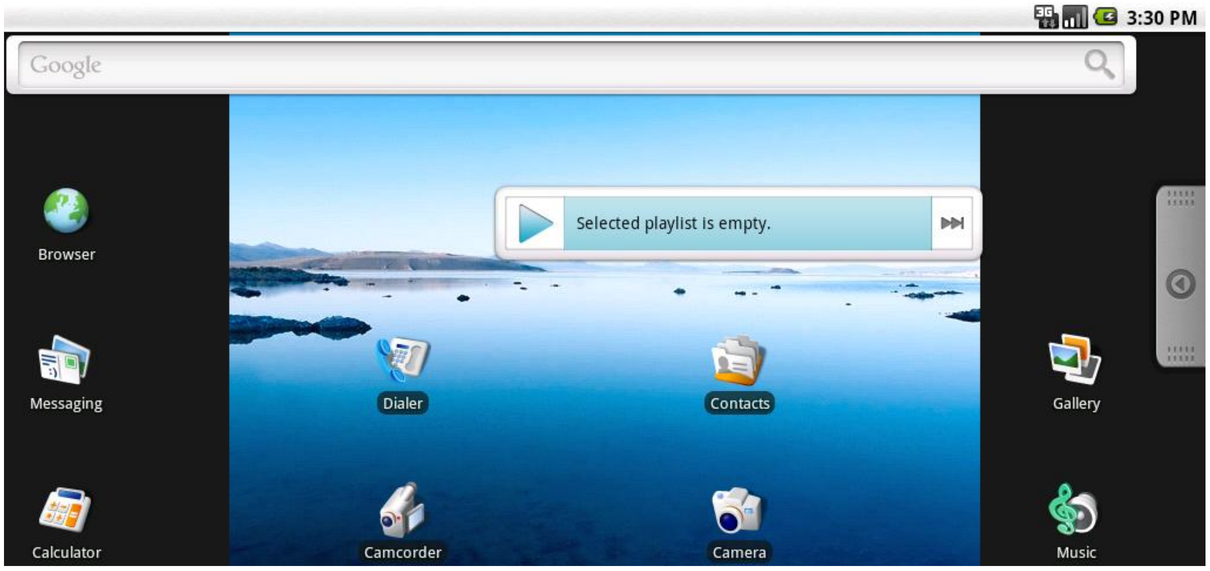
Modifying number of desktop workspaces

Android comes with a default of 5 home screens flickable from right to left and back. Mobile is a very personal device and Android being designed for such a device allows personalisation of various home screens based on user preference. A non-handset device could however be a shared device and such customization on various home screens may not be required. Based on the target device and use cases solution makers would probably want to either limit or increase the number of home screens. For a residential VoIP desktop device one home screen would suffice, where as a multimedia tablet device can still have multiple home-screens.

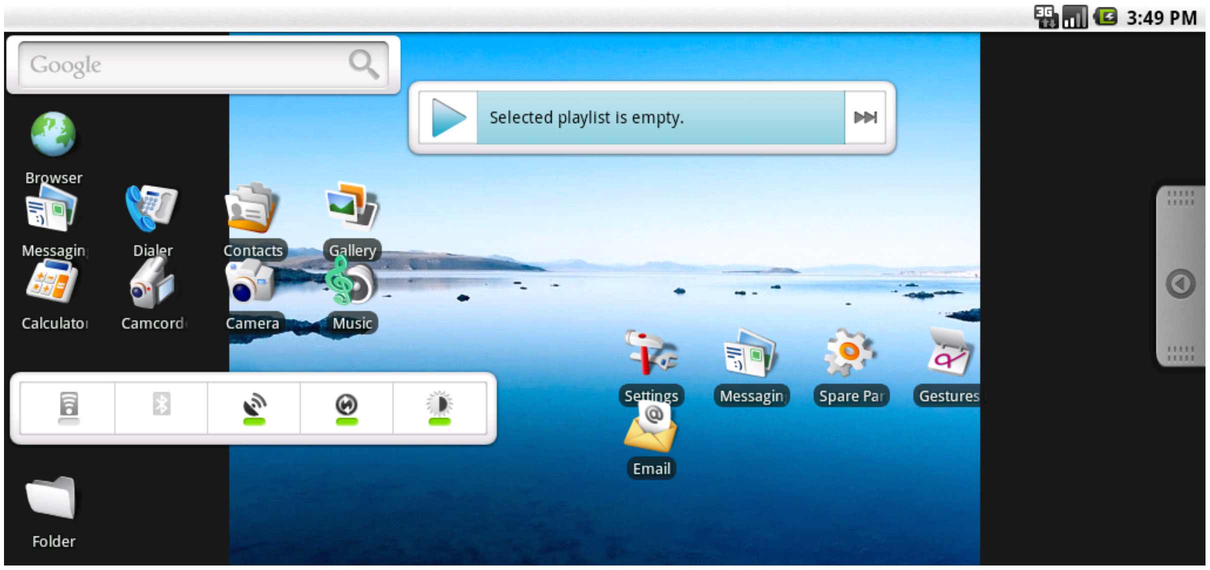
Customizations allow for personalization, but should be done within the UI framework to avoid messy merges during android version upgrade.

Modifying cell layout in launcher app

Android has a compelling widget framework, the ability to have relevant & active information right on screen without effort is hugely popular. Placement of these widgets on the home screen in terms of position and movement has been quantified into a mesh of 4 x 4 blocks effectively spanning the entire screen with 16 blocks. A 16 block grid is suitable sufficient for a smart-phone device with smaller screen dimension (not necessarily smaller pixels), but when stretched across a large screen the coarseness shows up in both movement and placement of widgets. Premium space on the screen deserves to be used more efficiently and finely. It makes sense to increase this grid mesh from 16 blocks to a reasonable number based on the screen size and processing performance available. To have a similar user experience when transitioned from 3.5" screen to a 9"screen, the number of blocks in the grid may have to be roughly increased by four folds.



1: *Default Android on 1024 x 480 screen with 4 x 4 grid mesh.*

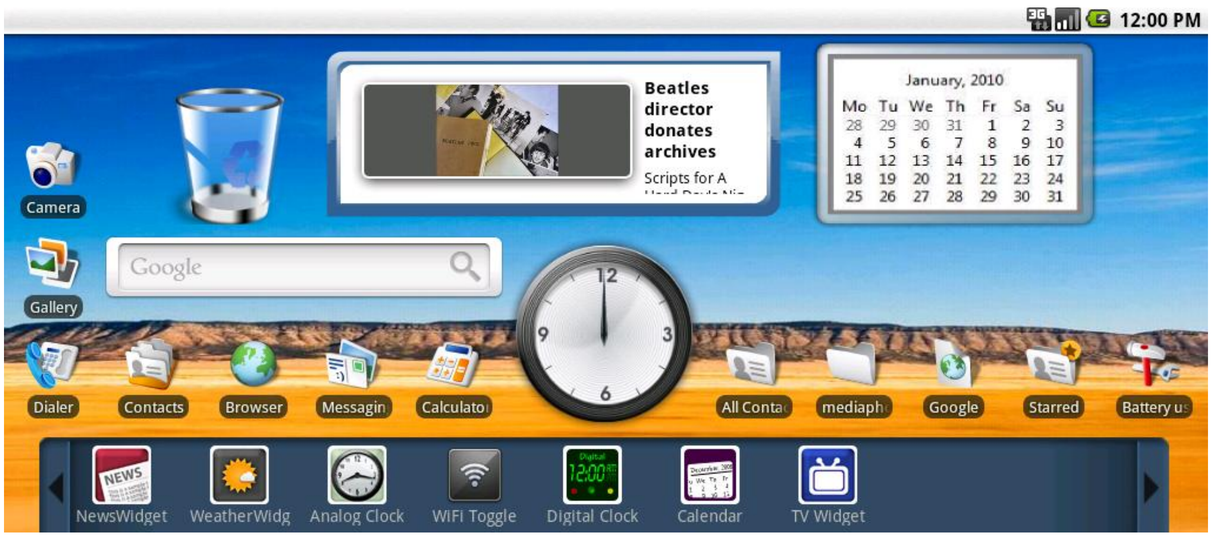


2: *Default Android on 1024 x 480 screen with 12 x 7 grid mesh.*

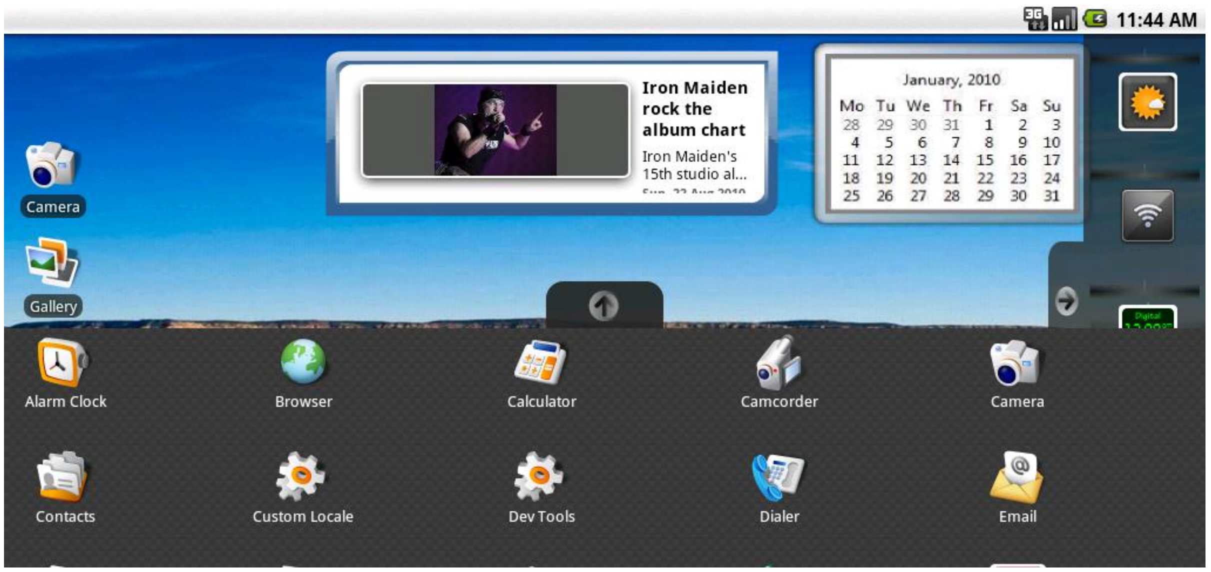
Cons: This change requires higher computational power and hence optimization of SKIA rendering engine or acceleration through hardware GPU becomes necessary.

Adding custom widget bar

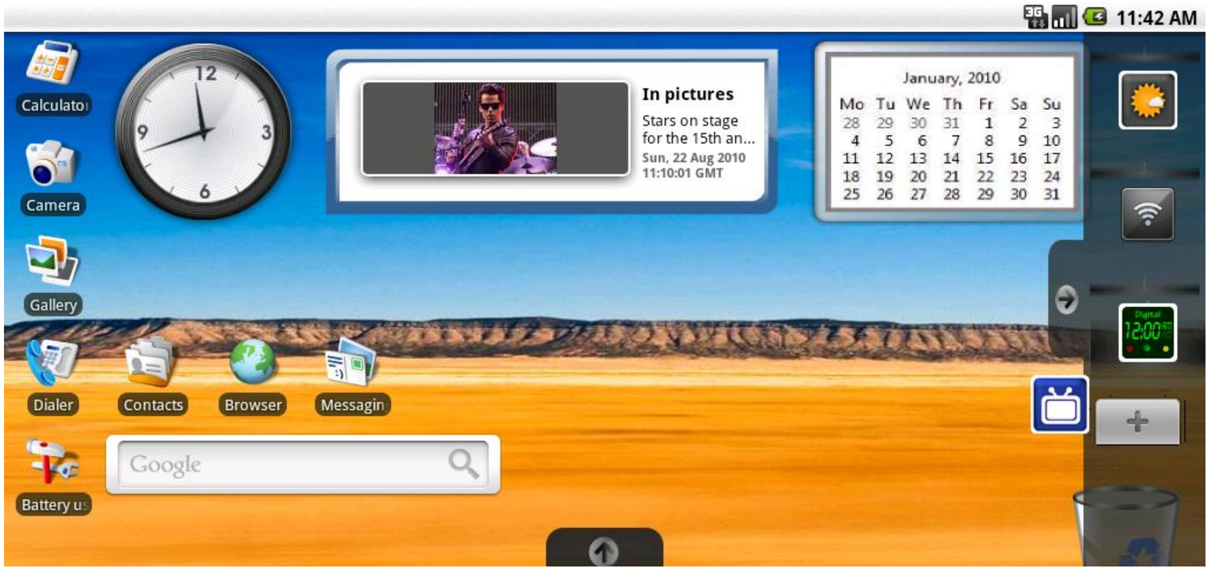
Instant information right on the screen defines characterises active widgets on the screen. If the user is made to go through a treasure hunt to find the widget of choice, it defeats the simplistic nature of having widget framework. Widget framework in Android enables users to pick and install from a wide array of available widgets. With widget framework in Android device makers or solution developers would write custom widgets using existing apps for their solutions. Users would choose which widgets to activate on screen and they would do well without having to be bothered about how to search for these widgets; it is a good idea to have custom widget launch bar which could list all or some of the available widgets on device. This launch bar simplifies usage of widget functionality in Android and thus makes it more suitable for wider range of embedded platforms. *It could even be a fixed set of widgets on the homescreen, suitable for devices like WebTV, Set-op Boxes etc., where user interaction is predominantly through a remote control device.*



3: Modified Homescreen with large number of app widgets and a custom widget launch bar

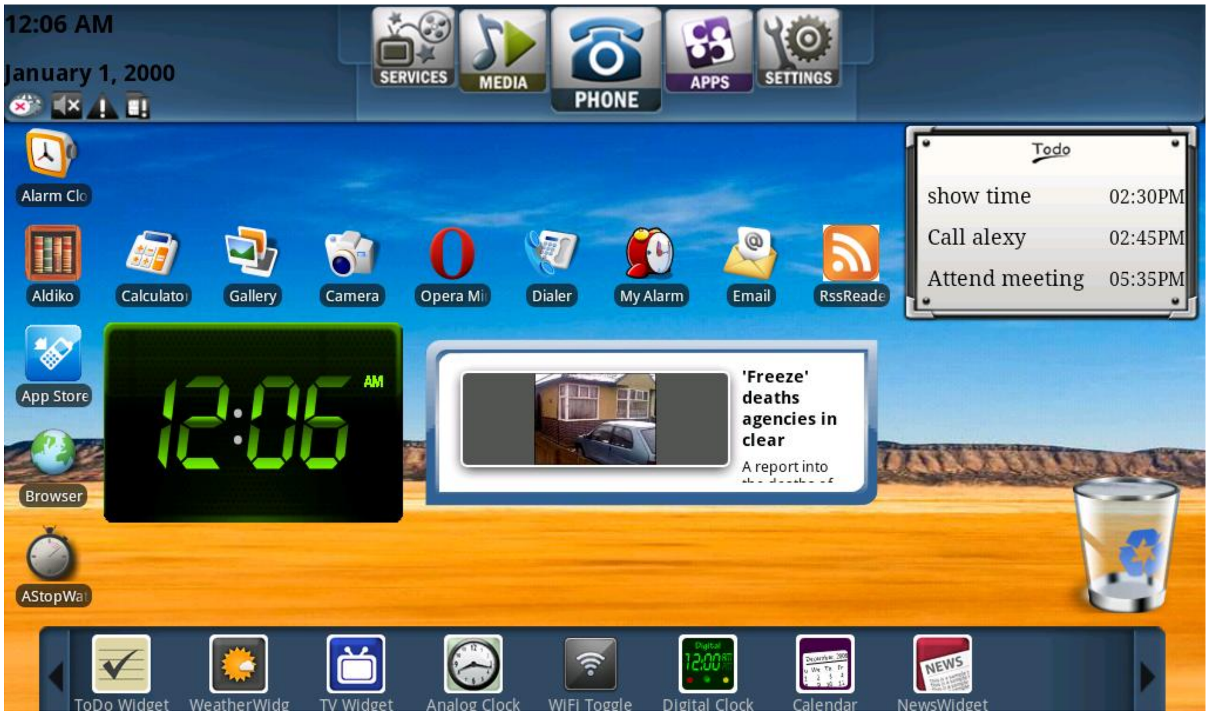


4: Default app sliding drawer widget can be placed from default right extreme to bottom, while a new custom widget launch bar added in its place.



5: *widget launch bar with functionality to add and delete app-widgets.*

Modifying Notification Bar
Notification bar in Android shows real time information about signal availability, battery conditions etc., On an always-plugged in non-wireless broadband based desktop VVoIP phone this real-estate could be better put to use by having tap-and-go buttons for most often used or priority applications.



6: *Customized homescreen with widget launch bar and modified notification bar*

Cons: This would require changes to be done in the core framework of Android. This also involves making changes for message passing between the framework and home screen application back and forth. Since this form part of the unexposed private APIs migrating to higher versions of android may be troublesome.

Conclusion

Popularity of android and its growth both within smart-phone market and outside, brings the need for device, model and OEM specific customizations. UI customizations within the published android SDK framework are necessary for future android version upgrades. Building custom backend services needs to be portable across versions and h/w platforms with minimal or no dependency on private or un-published APIs.