2020

# Jukebox: An Adaptive Collaborative Open Application Development Ecosystem

Ajay Mutreja
*University of Auckland*, ajay_mutreja@yahoo.com

Gabrielle Peko
*The University of Auckland*, g.peko@auckland.ac.nz

Johnny Chan
*University of Auckland*, jh.chan@auckland.ac.nz

David Sundaram
*University of Auckland*, d.sundaram@auckland.ac.nz

# Jukebox: An Adaptive Collaborative Open Application Development Ecosystem

## Completed research paper

**Ajay Mutreja**
Department of Information Systems and Operations Management
University of Auckland
Auckland, New Zealand
Email: ajay_mutreja@yahoo.com

**Gabrielle Peko**
Department of Information Systems and Operations Management
University of Auckland
Auckland Central, New Zealand
Email: g.peko@auckland.ac.nz

**Johnny Chan**
Department of Information Systems and Operations Management
University of Auckland
Auckland Central, New Zealand
Email: jh.chan@auckland.ac.nz

**David Sundaram**
Department of Information Systems and Operations Management
University of Auckland
Auckland Central, New Zealand
Email: d.sundaram@auckland.ac.nz

## Abstract

In this day and age products are expected to be delivered to the right consumer at the right time through the right channel, in a way that inspires, informs and excites them. An adaptive, collaborative, open, and flexible mobile application solution has been designed for implementation. This solution undertakes to promote a broad level of inter and intra industry inclusivity, increase efficiency and efficacy, reduce total cost of operations, secure high transactional revenues, optimally channelize investments for better returns with minimum data pilferage and stimulates value for the end-users. The solution uses a jukebox metaphor and proposes conceptual workflows, architectures, protocols, and an application development ecosystem. These artefacts are a call for standards that could potentially evoke a paradigm change in mobile app development and operations.

**Keywords** adaptive architecture, collaborative, open ecosystem, application development

# 1 Introduction

With users shifting to the mobile platform for consumption of their needs, the smartphone has emerged to be an important form factor. Smartphones support the prime objectives of engaging users, delivering undertakings and securing transactions. Apps are hosted on a platform and each platform such as Apple and Google have their own distribution commercials, guidelines, framework, and tools for development as well as hosting. This pushes businesses to incur expenditure in developing multiple instances of mobile apps for multiple platforms (Blair 2020). This necessitates expenditures in millions towards marketing for creating visibility, reach and intensifying user engagement. In return for this investment they get just a few seconds of users' screen time while the issues of app discovery, user acquisition, retention, and monetization still remain challenged. Despite all this expenditure, not all apps may be available on all platforms. On the user side, for installing the apps users may have to spend their precious time in searching, researching, downloading, installing and keeping the apps updated. Resulting in overcrowded smartphone storage and sparingly using them while they are kept running in the background. App activity on mobile devices consumes memory, data and the battery while also exposing security vulnerabilities. Further, multiple Apps running at the same time slows the device and causes lags in apps functionalities by overwhelming the operating system (OS), which leads to user dissatisfaction.

Considering these issues from the developer's perspective, issues of interest include: a) choosing the right app to develop b) the development technology c) different device and screen size compatibility d) dealing with different OS e) security and f) app distribution control (Bitmascot 2016). Based on this understanding of the perspectives of businesses, users and developers, our research question is: ***"What could be a better design solution for mobile applications?"*** A design that is able to engage users and secure their transactions without needing to be downloaded, installed, uninstalled and regularly updated. A design that does not consume too much of precious mobile phone resources (memory, battery, and data). A solution that is robust, secure and developable with lower expenditure yet works ubiquitously while able to dynamically link and concatenate with other apps to provide a customisable, adaptive, collaborative and open ecosystem.

This paper aims to answer the research question by utilizing the design science approach as elaborated in section 2. It goes back to the drawing board and reiterates the mobile application development design, redraws an end-to-end process to prescribe the solution, model, flow framework, architecture, protocol and implementation in a simplistic way. Section 3 is an update on the current industry norms of the mobile app development. Section 4 reviews the relevant App development frameworks. Next, Section 5 Analyses the current artefacts through a literature review and represent alongside, the new conceptualized artefacts, using the induction-deduction method. Section 6 offers implementation with a newly proposed protocol and Section 7 opens up the discussion for further research and concludes.

# 2 Methodology

Design is both a process and a product, Design science seeks to extend our boundaries by creating new and innovative artefacts to define the concepts, models, frameworks, architecture and systems (Hevner et al. 2004). And Design Science Research (DSR) is a set of analytical techniques for performing research in Information Systems involving two prime objectives, a) creation of new knowledge, and b) the analysis of the artefacts' use and/or performance with reflection as well as abstraction (Vaishnavi et al. 2019). In our research we used the DSR methodology for undertaking the cognitive process of getting aware of a problem, suggesting a solution, evaluating, analysing and reflecting upon the academic as well as the industry literature in the development of our artefacts. And finally detailing the implementation of our concept with the support of these artefacts. With low problem domain and solution domain maturity regarding our concept we may position "The Jukebox" concept with its "Piggyback Architecture", "Jukebox Application Development Ecosystem (JADE), "Glue Protocol" and "Yin/Yang wrapper" artefacts in the innovation domain.

# 3 Background

Several authors have conducted research in the field of mobile applications development as well as cross platform mobile application development frameworks, and some of the works reviewed are summarised in Table 1. The literature on the subject is broad, some papers present work on the design of a specific framework or its improvement with an explicit approach to development. Others argue that technological enrichment of a particular portion, address the subject through case studies, and comparisons looking at particularities of the element in a predefined context. However, it appears there

is a paucity of literature, in top tier IS journals in particular, which echo a solution similar to the one conceptualized and presented in this paper.

| **Authors** | **Work Subject** |
|---|---|
| Bernardes and Miyake 2016 | Cross platform mobile app development approaches. |
| Biørn-Hansen et al. 2017 | Progressive Web Apps. |
| Dalmasso et al. 2013 | Comparison and evaluation of cross platform mobile application development tools. |
| Dhillon and Mahmoud 2015 | Evaluation framework for Cross platform mobile application development tools. |
| Heitkotter et al. 2013 | Cross-Platform Model-Driven Development of Mobile Applications with Md2 |
| Hudli et al. 2015 | Selection of mobile app development framework. |
| Martinez and Lecomte 2017 | Quality Improvement of Cross Platform mobile applications. |
| Mahesh et al. 2012 | Portability of mobile applications. |
| Ohrt and Turau 2012 | Development tools for smartphone applications. |
| Palmieri et al., 2012 | Comparison of cross platform mobile application tools. |
| Sommer and Krusche 2013 | Evaluation of cross platform frameworks for mobile applications. |
| Stallings 2012 | Operating Systems, Internals and design principles. |
| Usman et al. 2017 | Model-driven engineering approach for generating feature based mobile applications. |
| Xanthopoulos and Xinogalos 2013 | Comparative analysis of cross platform development approaches for mobile applications. |

*Table 1. Authors and Their Work in the Field of Mobile Application Development*

Grey literature was also investigated to comprehend the emerging trend of super apps (Huang and Siegel 2019) providing the app-in-app solutions and building ecosystems by delving into insights from the cross-industry leaders such as KPMG (KPMG 2020), Amadeus (Amadeus 2020) and Accenture. Following the thread on one such ecosystem forerunner, we looked at the Chinese Super App, a proponent of Mini-Program framework. WeChat's developers' offerings on the syntax, components, APIs and other development tools were explored (WeiXin 2020). Skimming for similar tools offered by other super apps like Alipay (Alipay 2020), the works of Zapier on 'ZAPS' (Zapier 2020) was witnessed. And noted work on the progressing adoption of progressive web apps and resourcing understanding through sources such as Google developers' guide (Google 2019) on "Google Extensions", Microsoft documents on "MS Plugins", Salesforce's "Slack" Solution (Slack 2020), and plethora of articles.

# 4 Review

Before we go back to the drawing board for conceptualizing adaptive and collaborative systems, of all the frameworks mentioned in section 2 such as "ZAPS" and "SLACK" there are a few frameworks that are relevant to our design as baseline approaches. Grounded on their applicability of concatenated app performance, secured operations and non-compromised functionality while maintaining nativity, the three frameworks shortlisted for review are: a) app wrapping b) progressive web apps and c) mini-programs.

## 4.1 App Wrapping

Now a days almost everyone has a smart phone and the employees bring their phones to their work loaded with several applications, these sundry applications use office web network to communicate over internet, this poses a significant security risk of sensitive data disclosure and makes the company's network vulnerable to cyberattacks. App Wrapping or Application Wrapping is a process of adding a layer to the mobile app for management or security purposes without requiring any changes to the underlying app. It allows the administrator to set specific policy elements to any app operating within the controlled environment. Also, nowadays companies work with many value chain contributors and run expansive operations that require the admission of more people, devices and outside system connections to the expansive network. Businesses use app wrapping tools to operate this valued ecosystem, to share work, information and transact in a secure digital environment while conducting business with mitigated risks and threats. This mode of operation containerizes the sundry apps pooled into the ecosystem by the value chain service providers and offers a blanket of protection through security policies without changing anything to their individual applications (AppSolid 2017). The "wrapper" that encapsulates the container of these sundry apps, allows developers to set up container

appropriate usage policies to all the apps in the container with defined limitations to functions such as file-sharing and data storage. App wrapping is a very flexible mode of operation as policies can be adjusted any time and not much development work needs to be done on each individual app.

### 4.2 Progressive Web Apps

Progressive Web Apps (PWA) are www. applications that use website browser features to bring a native app like user experience to cross platform deployment. Majchrzak et al. (2018) mentions PWA as a definitive approach to cross platform app development. They promise to pool in the ease of development of web technology with the performance of the native apps. These apps are downloadable, installable, and uninstall-able just like native apps. PWA can get added to the home screen of user devices, can be partly used offline and are auto-update-able. They use the html framework but unlike website where a URL has to be submitted through the browser search window every time for interaction, it does not require this. PWA look like a regular app but works on a light version of the browsers in the background where all the browser interface features such as address bars and menus are hidden. As per Google Play Developer API guide (Google 2019), the PWA content such as HTML, CSS, JAVASCRIPT, FONTS, AND IMAGES are added to the user device at the first visit, unlike websites, users do not have to wait for all the content to be downloaded on every visit. Compared to the Native Apps PWA leave the least footprint, use less of user's device resources, and since it is web based, it implements a responsive design to work satisfactorily across devices of different screen sizes and resolutions. The PWA's are just like browser websites, the interaction with them is primarily online with the requirement of availability of internet at all times. However, PWA are not currently supported by the Apple's Safari browser and don't run on iOS devices making them non-universal or non-ubiquitous.

### 4.3 Mini-Programs

A Mini-Program (MP) is a little downloadable program with small functionalities of the main application that run inside another larger applications. MPs such as WeChat and Alipay are very popular in China and may be take over IOS and android app ecosystems therein, they essentially operate as a separate mobile app, except that they function within another larger app. They allow users to use the apps they need without leaving the main app. They are very light therefore called Mini, and there is no need to download them to the device, can be accessed anywhere, anytime, on any device and if no longer needed, can be simply unpinned from the larger app. The deleted MP can be re-accessed when needed again by simply searching the MP in the directory and clicking them to pin them to the larger app with no download and no reinstallation. MP use Single Sign On (SSO) credentials of the larger app and also use the device resources through the larger app but is cursed with the walled garden syndrome. This syndrome generates extra work for developers when creating MP for various platforms such as WeChat and Alipay, which promote propriety web development tools for use only in their respective environments (WeiXin 2020, Alipay 2020).

## 5 The Jukebox Concept

In the early 20th century, a Jukebox was a music playing device that assembled the components together to play music on demand from the music tracks stored on vinyl records in a self-contained media placeholder and allowed the vinyl to be easily mount, unmount and remount on the operating turntable to play music as per user selection. Using the analogy of 'Jukebox' our concept facilitates the mounting, unmounting and remounting of the mobile applications on the operating host application for delivering the functionality on demand and as per user selection. With objectives to reduce complexity, increase ease of use, decrease associated development cost, deliver best experience and be ubiquitous. We aim to propose a best-of-breed framework, and deduce our conceptual solution artifacts for implementation. We first introduce and present a literature review on three critical elements of the mobile app design and development, i.e. a) the workflow – to understand the end to end app usage, its deliverables and related process flows for design and configuration of mobile apps, b) the architecture – to describe the elements of the design and the structure of the arrangement of those elements in development of the apps, and c) the application development framework – to support mobile application development in accordance to its architecture. And alongside we propose our novel artifacts "The Jukebox workflow", "the Piggyback architecture", and the "Jukebox application developmental ecosystem (JADE) ".

### 5.1 The Jukebox Workflow

Workflows are defined as a series of events that can be simple or complex, can run in sequence or in parallel, and must have specified rules for action (Smartsheet 2016). Just like a scientific process where the same workflow delivers the same result every time it runs. A workflow consists of an order of steps,

having a starting point, middle action points, decision points, and ending points. Resources are needed at each step. There are rules of operation for these steps, it has a direction of movement, expected outcomes and potentially have substitute workflows or steps. Further, workflow is empirical and can be measured. Successful workflows can help improve communication with and within the participants and reduce user interface. Many workflow management systems integrate existing otherwise independent systems into a unified system and are capable of automating the workflows by organizing resources from multiple sources for automated routing and processing. They can form inter-process pipelines; can give the next process the data required and provide notifications for completed and uncompleted steps. These capabilities save time and effort, increase efficiency, and thus adds a lot of value to the system design. 'ZAPS' by Zapier (Zapier 2020) and 'Slack' by Salesforce (Slack 2020) are dedicated to streamlining workflows with pre-defined API connections between the Applications. Users can browse for the applications that are pre-connected and unify the workflows between these otherwise independent applications as needed, for short or long-term use. For example, you can define a workflow such as an e-mail with a flight reservation that arrives in a g-mail mailbox. The attachment is stored in the drop box cloud storage and the schedule date and time is updated on google calendar to trigger a reminder notification. The users are charged based on the inter-system workflow instance count meter and as per subscription contracts between the user and the service providers. It's a B2C offering with pre-contracted B2B agreements between the platform and the otherwise independent systems/applications and the service subscriber.

Using a scenario from the travel industry where travelers consume several travel services such as flights, hotels, transportations, translations, payments and so on. To do so, these travelers need to download and use several mobile apps during their journey from their home country to the host country and back (Amadeus 2015). Some of these apps are ubiquitous, such as frequent flyer apps, and therefore can be used for all journeys while others have to be deleted upon returning home and others have to be added before embarking on a new journey to a different destination. The scenario can be mapped as in Figure 1, where each service component action is delivered by a separate app, there is no streamlined work flow through API connection between these apps.

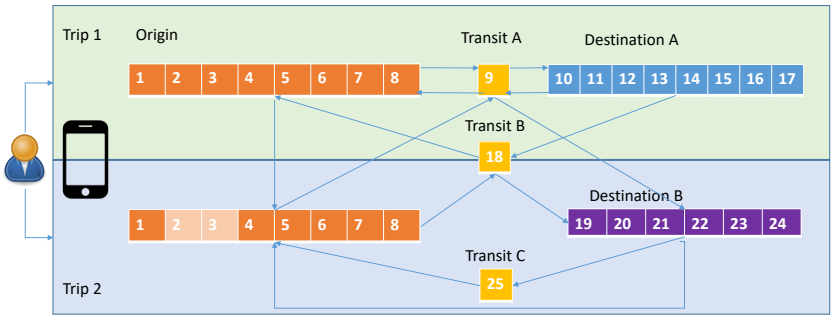



*Figure 1: App download and usage scenario at present*

Alternatively, imagine the same scenario where just one app is downloaded, installed and used in the foreground of the device by the user (the host app) and the work flows to various independent apps (guest apps) through the host app, without any need of downloading, installing, uninstalling and updating the guest apps. The reimagined scenario with a new convenient workflow 'the Jukebox' is represented in Figure 2.

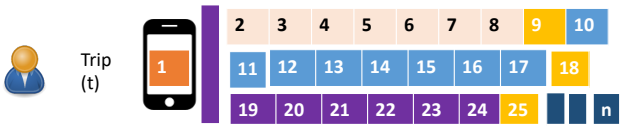



*Figure 2: Jukebox representation*

The Jukebox workflow has been portrayed with a Data Flow Diagram (DFD) in Figure 3. The Jukebox workflow involves eight steps, by which the user visits the Jukebox App directory from within the host app, such as Google Maps. The host app will raise a query to the app directory for compatible apps and based on the query the glue-able guest apps will be displayed. Users select the apps to glue them with SSO credentials. The glued apps will be available in the jukebox container menu bar for easy navigation, use, detachment and reattachment.

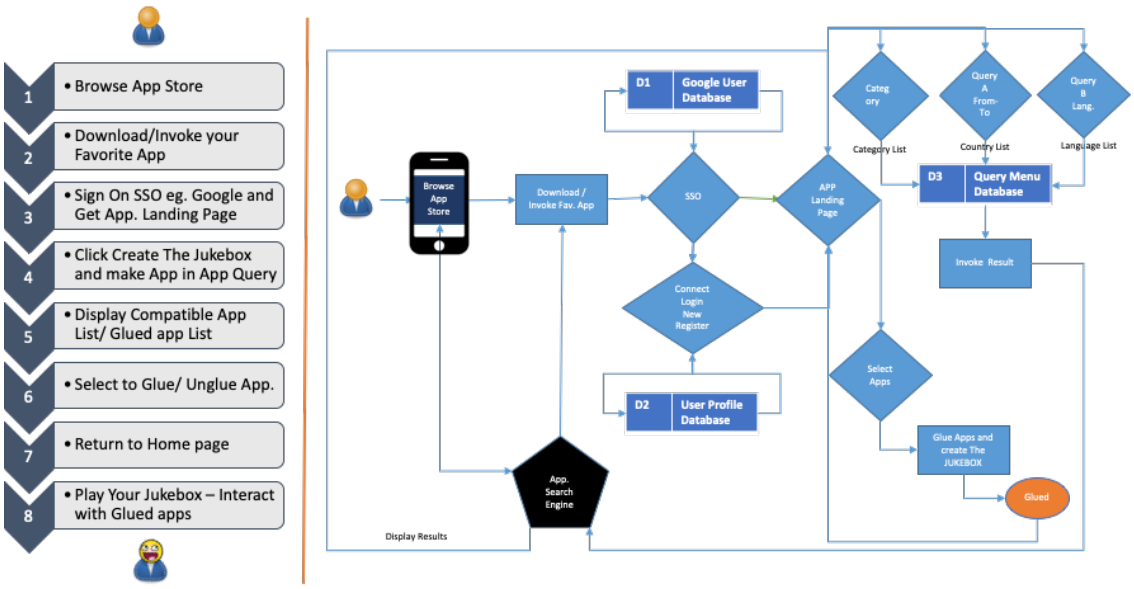



*Figure 3: Jukebox data flow diagram*

Based on the above understanding of the workflow the Jukebox Wireframe has been crafted and displayed in Figure 4.

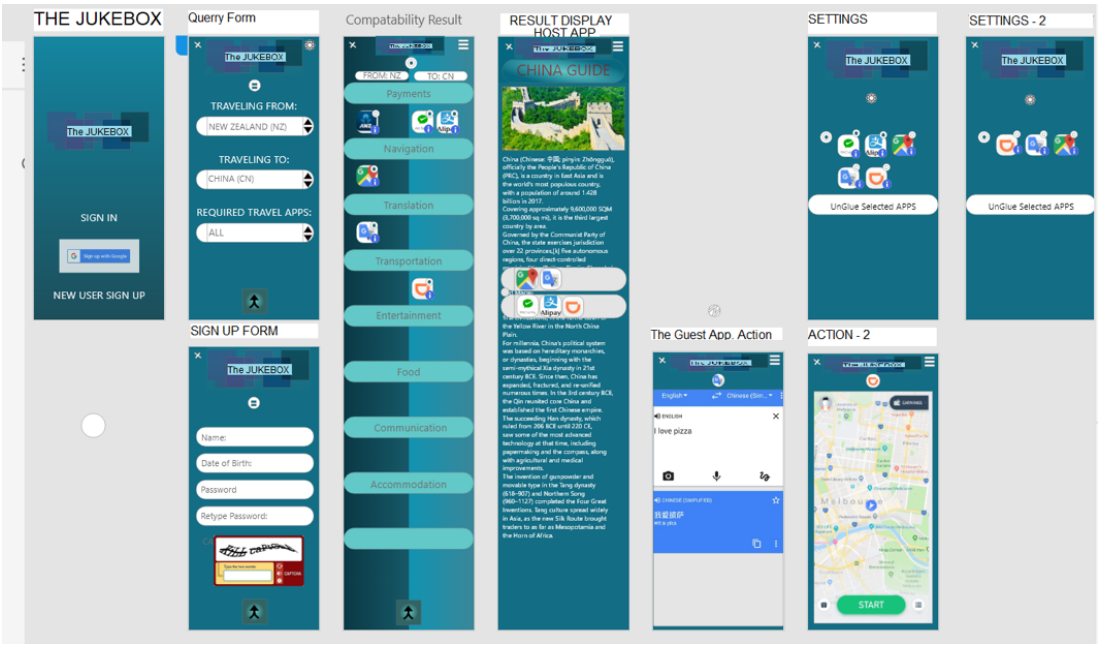



*Figure 4: Jukebox wireframe*

## 5.2 The Jukebox Piggyback Architecture

Bachmann et al. (2010), defines architecture as a fundamental structure of a system comprising of the elements, relationships among these elements, and the elements' properties. The purpose of an OS architecture is to define and support the structure, behavior, and views of the system that runs the software on the device by providing services to the programs (Stallings 2012). OS act as an interface between mobile applications and the mobile device hardware. There are several layers in the smartphone OS architecture such as the Android: kernel, libraries, application framework and applications (Sharma 2019). When users use a mobile device, such as their smartphone, they typically perform several tasks at the same time. This is called multitasking. To provide for seamless multitasking, the OS coordinates the activities of the processor, it uses RAM as a temporary storage area for instructions and the data the processor needs. It manages the secondary storage (such as SSD, HDD) by upholding a file management scheme that keeps track of all the names and the locations of the files and the programs. Programs, called device drivers, facilitate the communication between the OS and the hardware components

housed in the smartphone estate such as battery, mic, speaker, and camera. Device drivers are included in the OS software and translate the commands of devices to commands that the OS can understand, and vice versa, enabling the OS to communicate with every housed device. All applications need to interact with the CPU, for which they must contain instructions the CPU recognizes. Instead of each software application carrying the same set of instruction and the CPU rendering the same information repeatedly from multiple applications, the OS includes this standard block of codes as an interface to which each software application refers to. These interfaces, called application programming interfaces (APIs). So, all that any application needs are the corresponding API code to interact with the operating system. Then the OS will maneuver the device resources as necessitated by the application. Hence, for an app to work it has to communicate with the OS through a kernel using libraries and runtime libraries. This logically signifies that Android runtime libraries can be used on other non-Android devices, as long as the OS on those devices can interpret them. It is thus possible to have one standard design for all mobile devices.

There are two prevalent mobile device OS available today, Apple's iOS, and Google's Android, both have their own set of libraries and both of them take different approaches to the mobile operating system. Apple distributes the devices that only natively support iOS and it takes a walled garden approach in which Apple regulates all mobile apps and services that can run on the iOS devices to run on its own XNU kernel, Google takes an open-source approach, this means that mobile device manufacturers can customize the Android source code and customize it to fit their devices to run on the Linux kernel. Further, the mobile device market is divided between Apple and Android, for a complete market reach business have to spend twice the amount of money to develop two separate applications for two separate OSs. Many big, deep pocket companies like Facebook, Line, Tencent, Huawei, and Alibaba are all following a type of walled garden approach creating their own run time libraries, development tools, development languages and promoting their own ecosystems further causing pressure on the developers to develop and redevelop multiple instances of their applications for each one of these platforms. For our concept to work, the application has to work platform independently, ubiquitously such that one app should be able to run on all OS (AppVelocity 2019; Helios 2017). Though it may be possible for users to jailbreak devices through their root directories, which allows them to install runtime libraries to run other mobile OS and unlock restricted applications, it is a complicated process for a layman and kind of illegal. Our solution ought to favor legality and ease of use, therefore we propose a Piggyback Architecture in Figure 5. In this architecture, the host app can be customized for several ecosystems and the guest apps need to be developed only once. Also, these guest apps would ride on the host app to operate on any platform as good as the native one.

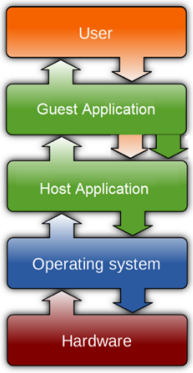



*Figure 5: Overview of the Jukebox Piggyback Architecture*

Figure 6 presents a detailed view of the Piggyback architecture. With our solution implementation we do not wish to compromise any of the app functionalities and user experiences so the apps should remain natively stronghold in its application architecture for highest possible performance. Yet, run seamlessly on all devices while consuming the minimum device resources as possible and performing auto-updates as and when necessary. For this purpose, we propose the use of a wrapper to wrap up the applications that can be glued together to work together as in Figure 6. The wrapper shall clamp security, connection, communication and update features while maintaining the intact presentation, business, and data layer integrity.

## 5.3 The Jukebox Application Development Ecosystem (JADE)

A mobile application development framework is a fundamental software structure that is designed to support mobile app development in accordance with its architecture for system specific environments.

Majchrzak et al. 2017, presented frameworks in three categories: 1) native framework, 2) mobile web-based apps framework, and 3) hybrid apps framework or cross-platform development framework.

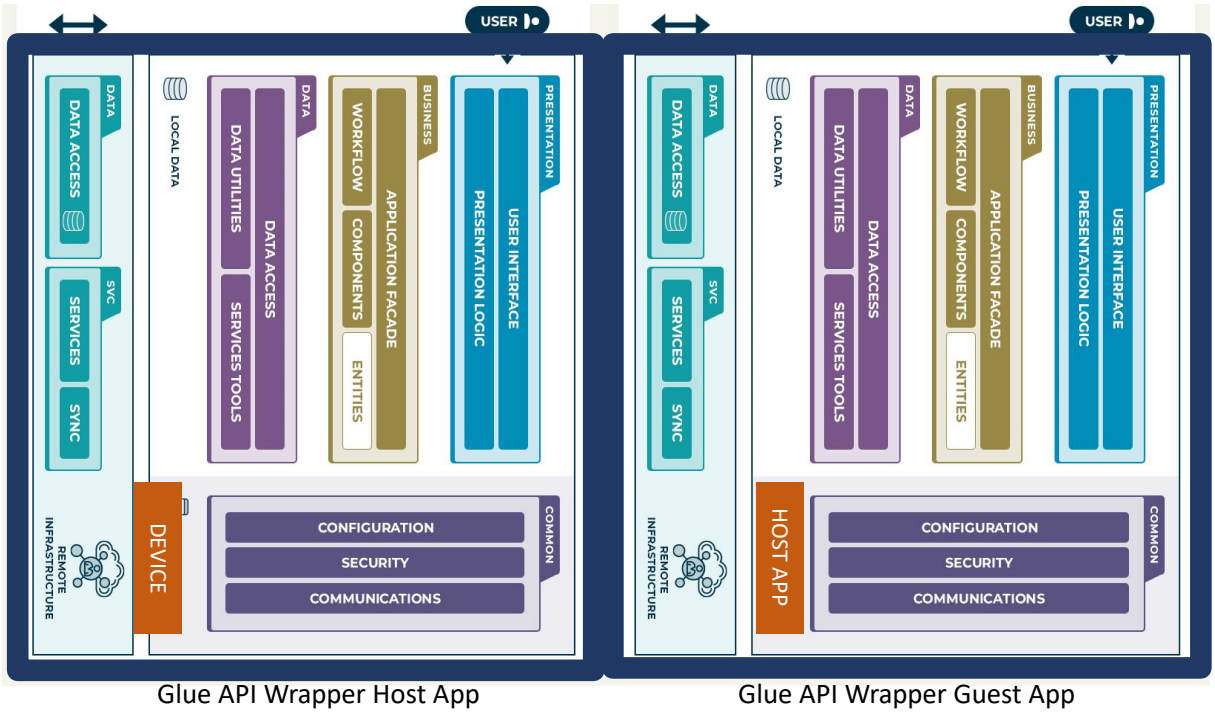



*Figure 6: Detailed view of the Jukebox Piggyback Architecture with wrapper.*

All apps targeted toward a particular platform are known as native apps. Therefore, apps intended towards Apple devices use the development tools specific to Apple devices, they do not run on Android devices and similarly the apps intended towards Android devices do not run on Apple devices. While developing native apps, for better performance, consistency, and user experience, the developers use the best-in-class user interface modules. This includes wider access to native API accounting for limitless use of all apps from the particular device. However, what is needed is the best performing universal app to run independent of the OS, to be developed with standard APIs instead of Native APIs and performs as functionally as native apps.

Since all the personal databases are saved on internet servers, users can fetch their desired data from any device through the internet using web-based apps. Web apps run on a web server and are not stored locally on the device OS; therefore, they are light and the updates need not be performed locally on all devices. Once the update is done at the backend users always access the up-to-date applications. PWA are the evolved version of the web apps, they are developed using this framework and have been introduced in section 3.3. This partially provides the solution but requires a browser to browse the apps and continuous internet access is required. This means applications cannot work offline and would suffer browsing buffering lagging, therefore, they do not provide a user experience as good as the native app. Also, as mentioned in 3.2, PWAs' are non-ubiquitous.

As the name suggests a hybrid app is a mix of native and web-based apps. These are designed to support native developments for multiple platforms with one code base, therefore they are easier and faster to develop. Cross-platform app development tools are used to create a single code base that auto-create multiple runtime libraries for publishing on multiple platforms. A developer writes the program code only once, however, multiple instances are published as per the standards defined by various platforms (currently supports only iOS and Android). Despite such advantages, hybrid apps exhibit lower performance. Often, hybrid apps fail to bear the same look-and-feel in different mobile operating systems and apps still need individual downloading, installing, uninstalling and updating. Also, consumption of device resources is not improved plus specialized development tools need to be used.

The Jukebox in itself is an app that contains other apps. We call these Jukebox apps, JAPP. For developers, there would be two types of apps, the host JAPP and the guest JAPP. The host JAPP are developed using platform specific framework configured additionally with a standardized host side wrapper to facilitate the gluing of the guest JAPP. A single instance of the guest JAPP is developed using any tool and shall configure a standardized guest side wrapper, as shown in Figure 6. Every guest JAPP

shall be available in the JAPP Store, and is searchable through the search functionality embedded in the host JAPP connecting to the standardized JAPP Store directory. Users shall search JAPPs from JAPP Stores and based on the query parameters get the results for glue and play, the JAPPs would support automated workflows and connect, disconnect, reconnect spontaneously, using standardized APIs & Protocols. Mutualism policies among them would restrict and permit what applications can share with each other. There would be some additional development efforts required at the host JAPP side to orchestrate these policies and render a placeholder for the guest JAPP however these efforts are just one-time efforts for all the potential glue and play. It is important to note that for the above workflow we would need, a standardized JAPP Store, a standardized JAPP Store search engine and a standardized JAPP gluing protocol. This allows guest JAPPs to piggyback on host JAPPs, which are accessible through the host JAPP environment, and draw kernel and other resources through them.

If this Jukebox Application Development Ecosystem (JADE) is standardized it would render the development of all apps as compatible to drive convenience, cost effectiveness, consistency, ubiquity, performance and adoption. The business would have to strategize the core operational strategy, which would define whether the app to be developed should be developed as a host JAPP or a guest JAPP based on the function of the app to be purely transactional or to be an ecosystem. Figure 7 demonstrates, how various app development frameworks can be placed on a matrix of three parameters: platform independence, performance, and mobile resource dependence. In comparison to other prevalent frameworks, it is depicted that with desirous positioning of JADE in top right quadrant signifies that, "JADE' would support high platform independence, relating to the ease of use, as the app aims to deliver the same functionality with the same look and feel over different platforms, it would reduce complexity of working on several platform (native or hybrid) development tools, reducing associated development cost as well as increasing ubiquity. Low mobile resource dependence is related to best experience, and high performance warrantying the best in class solution.

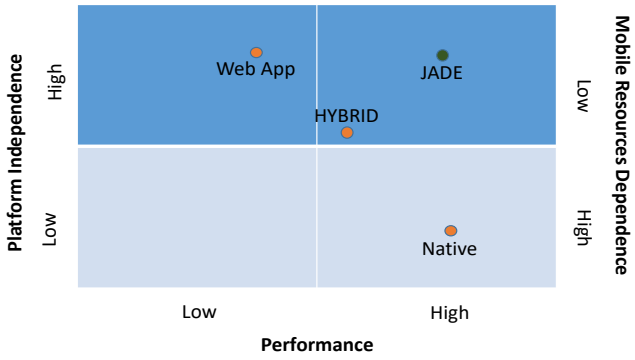
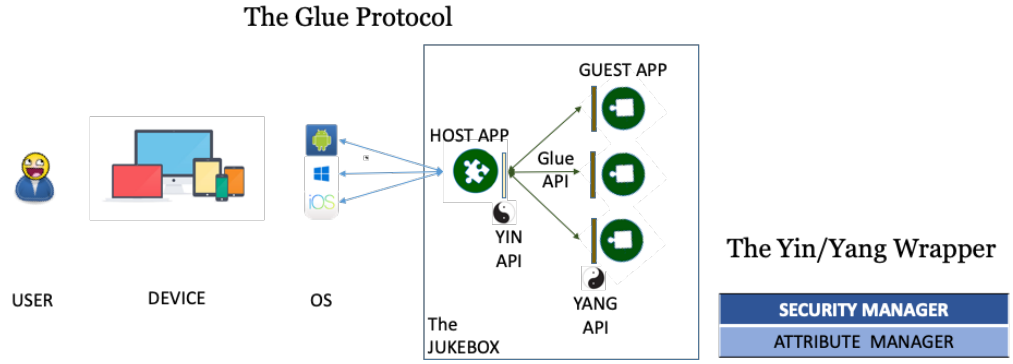



*Figure 7: Evaluation of Application Development Ecosystems*

# 6 The Jukebox Implementation

If we take the proposed Mini-Programs (MP) approach, standardize it, and govern it with universally accepted standardized protocols it has the potential to become a new paradigm for the application market which would dissolve wall gardened ecosystems. Giving equal opportunity to all enterprises to collaborate, co-create new values and provide a standardization for the developers. This is what 'The Jukebox' aspires to achieve. Referring to the analogy of Jukebox in section 4, in our Jukebox, the host JAPP act as a turntable and the guest JAPP act as the vinyl records. To standardize the adoption, the Glue Protocol is an arm to handle the play. According to the Glue Protocol of the Jukebox (Figure 8), the host JAPP side wrapper is called the Yin wrapper and the guest JAPP side wrapper is called the Yang Wrapper. These wrappers are a set of standardized corresponding API's on the respective side of the applications, analogous to Velcro adhesive, and used to perform the security handshake, connect and communicate. It would work in such a way that when the JAPPs get glued a secure connection is established. The Yang Wrapper API communicates with the Yin Wrapper API, which triggers the embedded original host app API to communicate with the device OS. A tunneled connection is established between the device OS and the guest JAPP through the host JAPP. The resources of the device, such as the battery, memory, storage, and camera, are accessed by the guest JAPP through the host JAPP. The host JAPP in this case becomes the secondary OS or a virtual OS on top of the device OS and executes the Piggyback Architecture as seen in Figure 6. In this arrangement, the host JAPP would draw and consume more resources from the device. But the device OS would not be overloaded with multitasking over multiple apps as resources are only operated upon by the host JAPP. Also, there are

no local device DIU2 (downloading, installing, updating and uninstalling) of the guest JAPP and no security threats from them. Nonetheless, multithreaded communication between the OS and host JAPP should provide the speed and processing benefits to the guest JAPP.




The Glue Packet

| 8 Bit | 32 Bit | 8 Bit | 8 Bit | 0-296 Bit | 24 Bit |
|---|---|---|---|---|---|
| Preamble | Access Address | Header | Length | Data | CRC |

*Figure 8: The Glue Protocol*

Two corresponding parts are proposed to be on each side of the wrapper as shown in Figure 8. The security manager defines a simple protocol that provides a security tool for generating hashes of data, generating confirmatory values, and generating short term keys used during gluing. Gluing is typically by host JAPP as an authenticator and guest JAPP as a requestor, followed by encrypted linking and key exchange so that the JAPPs can securely interact and, if required, can easily reconnect at a later date. Keys management may be governed by standardized Public-Key Cryptography Standards. The Attribute Manager, defines a set of rules of communication for the exchange of data and executing functionalities between the gluing apps such as API definitions and resource governance credentials. While binding, the wrappers may exchange information in the form of packets to get glued, unglued and re-glued. The packets may be of variable length, they include the host/guest identifying information preamble, address, content description, app payload, data freight, and a checksum to govern and validate every interaction as in Figure 8.

# 7 Discussion and Conclusion

In order to address the research question, this paper presents work from academia and industry in several focal areas. Several exploratory elucidations were introduced and their underpinnings to conceptualize the solution that, theoretically, can dynamically link and concatenate mobile applications into an adaptive, collaborative and open ecosystem. The artifacts presented here reduce complexity and associated development cost by foregoing the need to develop multiple instances of the same application for several platforms using multiple platforms' native and/or hybrid toolboxes. Converting or developing an app to be a guest app, that can be deployed with equity on several host platforms make them ubiquitous. Ease of use and device resources optimization is promoted by reduction in Downloading, Installing, Updating and Uninstalling operations. And non-compromising nature of the architecture promises to deliver the best experience with full app functionality.

The Jukebox workflow, architecture, application development ecosystem and implementation protocols could become a governing standard to facilitate a great degree of sovereignty to developers as well as users. Developers would need to develop only standard apps that would ubiquitously work on all platforms, which users could glue, unglue, re-glue to play as and when they need, where they need, and how they need, without expansive DIU2 ops. and device resource consumption.

The relationship between the host and the guest app in the JAPP is based on the principle of mutualism. The guest gets the balanced resource, quality, and cost advantages over multiple platforms while the host app is able to harness the ecosystem advantages, traffic engagement, and maximization. Together they can serve as a comprehensive tool driving higher user satisfaction, utilization, acquisition, and transaction thus motivating cross-industry adoption. Working together they serve a great tool for the generation of comprehensive big data for the industry. The traffic from all the glued application

tunneling through one channel may solve the big data collection challenges to make the recommender and personalization systems even more effective furthering the relevancy of the solution. Conversely, the proposed solution may pose a threat to the profits that some walled gardened platforms generate. This threat to the businesses bottom lines may generate resistance to support the adoption of such a universal solution. However, this polarizing behavior and the fragmentation of the platforms are better envisaged as temporary hurdles in moving towards an adaptive, collaborative, and open application development ecosystem. Standardization and governance of the universal glue protocol has the potential to become a new paradigm for mobile application development which could liquesce the wall gardened approaches.

# 8 References

Alipay. 2020. "Mini Program | Services & Tools | Documentation." (https://global.alipay.com/docs/ac/tool/miniapp, accessed May 10, 2020).

Amadeus. 2015. "The Rise of the Mobile-App Empowered Traveler," *Insights/Travel Industry Trends*, , March 13. (https://amadeus.com/en/insights/blog/infographic-the-rise-of-the-mobile-app-empowered-traveller, accessed November 3, 2020).

AppSolid. 2017. "What Is App Wrapping?", *What Is App Wrapping?,* October 5. (https://blog.se.works/what-is-app-wrapping, accessed June 10, 2020).

AppVelocity. 2019. "The Ultimate Guide to Mobile Application Architecture." (https://www.appvelocity.ca/blog/guide-mobile-application-architecture, accessed July 1, 2020).

Bachmann, F., Bass, L., Clements, P., Garlan, D., Ivers, J., Little, M., Merson, P., Nord, R., and Stafford, J. 2010. *Documenting Software Architectures: Views and Beyond*, (Second.), Addison-Wesley Professional.

Bernardes, T. F., and Miyake, M. Y. 2016. "Cross-Platform Mobile Development Approaches: A Systematic Review," *IEEE Latin America Transactions* (14:4), IEEE, pp. 1892–1898.

Biørn-Hansen, A., Majchrzak, T., and Grønli, T. M. 2017. " *Progressive Web Apps: The Possible Web-Native Unifier for Mobile Development", 13th International Conference on Web Information Systems and Technologies (WEBIST 2017)*, SCITEPRESS, pp. 344–351.

Bitmascot. 2016. "Top 10 Challenges Faced by App Developers." (https://www.bitmascot.com/top-10-challenges-faced-mobile-app-developers, accessed July 1, 2020).

Blair, I. 2020. "Mobile App Download Statistics," *Buildfire*. ((https://buildfire.com /app-statistics, accessed May 10, 2020).

Dalmasso, I., Datta, S., Bonnet, C., and Nikaein, N. 2013. "Survey, Comparison and Evaluation of Cross Platform Mobile Application Development Tools," *2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC)*, pp. 323–328.

Dhillon, S., and Mahmoud, Q. H. 2015. "An Evaluation Framework for Cross-Platform Mobile Application Development Tools," *Software: Practice and Experience* (45:10), pp. 1331–1357.

Google. 2019. "Google Docs," *Google Play Developer API*. (https:// developer.android.com/google/play/developer-api, accessed May 10, 2010).

Heitkötter, H., Majchrzak, T. A., and Kuchen, H. 2013. "Cross-Platform Model-Driven Development of Mobile Applications with Md$^2$," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, SAC '13, New York, NY, USA: Association for Computing Machinery, pp. 526–533.

Helios. 2017. "Why Mobile App Architecture Is Vital for Mobile App Development." (https://www. Helios solutionsco/blog/mobile-app-architecture-vital-mobile-app- development, accessed May 10, 2020).

Hevner, A. R., March, S. T., Park, J., and Ram, S. 2004. "Design Science in Information Systems Research," *MIS Quarterly* (28:1), Management Information Systems Research Center, University of Minnesota, pp. 75–105.

Hudli, A., Hudli, S., and Hudli, R. 2015. "An Evaluation Framework for Selection of Mobile App Development Platform," in *Proceedings of the 3rd International Workshop on Mobile Development Lifecycle*, MobileDeLi 2015, New York, NY, USA: Association for Computing Machinery, pp. 13–16.

Ionescu, V. 2016. "Using Cross Platform Development Libraries. Telerik Mobile," *2016 15th RoEduNet Conference: Networking in Education and Research*, pp. 1–6.

KPMG. 2020. "Super App or Super Disruption?" *KPMG Global*, January 13. (https://home.kpmg/xx/en/home/insights/2019/06/super-app-or-super-disruption.html, accessed June 1, 2020).

Mahesh, B. R., Kumar, M. B., Manoharan, R., Somasundaram, M., and Karthikeyan, S. P. 2012. "Portability of Mobile Applications Using PhoneGap: A Case Study," in *International Conference on Software Engineering and Mobile Application Modelling and Development (ICSEMA 2012)*, December, pp. 1–6.

Majchrzak, T. A., Biørn-Hansen, A., and Grønli, T.-M. 2018. "*Progressive Web Apps: The Definite Approach to Cross-Platform Development?"*, January 3.

Majchrzak, T., and Grønli, T.-M. 2017. *Comprehensive Analysis of Innovative Cross-Platform App Development Frameworks*, January 4.

Martinez, M., and Lecomte, S. 2017. "Towards the Quality Improvement of Cross-Platform Mobile Applications," in *2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, Buenos Aires, Argentina: IEEE, May, pp. 184–188.

Ohrt, J., and Turau, V. 2012. "Cross-Platform Development Tools for Smartphone Applications," *Computer* (45:9), pp. 72–79.

Palmieri, M., Singh, I., and Cicchetti, A. 2012. "Comparison of Cross-Platform Mobile Development Tools," *2012 16th International Conference on Intelligence in Next Generation Networks*.

PWA. 2020. "Progressive Web Apps." (https://web.dev/progressive-web-apps, accessed June 1, 2020).

Sharma, R. 2019. "Developing for Android," *Developing for Android*. (https://www.cprogramming.com/android/ android_getting_starte, accessed July 1, 2020).

Slack. 2020. "Resources Library," *Slack*. (https://slack.com/resources, accessed May 1, 2020).

Smartsheet. 2016. "Essential Guide to Workflow Management", August 25. (https://www.smartsheet.com/save-time-taking-time-creating-workflows, accessed July 1, 2020).

Sommer, A., and Krusche, S. 2013. "Evaluation of Cross-Platform Frameworks for Mobile Applications," in *Software Engineering*.

Stallings, W. 2012. *Operating Systems: Internals and Design Principles*, (7th ed.), Pearson: Prentice Hall. (cuuduongthancong.com).

Tian, L., Du, H., Tang, L., and Xu, Y. 2013. "The Discussion of Cross-Platform Mobile Application Based on Phonegap" *2013 IEEE 4th International Conference on Software Engineering and Service Science*.

Usman, M., Iqbal, M. Z. Z., and Khan, M. U. 2017. "A Product-Line Model-Driven Engineering Approach for Generating Feature-Based Mobile Applications," *J. Syst. Softw.*

Vaishnavi, V., Kuechler, W., and Petter, S. 2019. *Design Science Research in Information Systems*, AIS. (http://www.desrist.org/design-research-in-information-systems/).

WeiXin. 2020. "Miniprograms," *Mini-Program/Documents*. (https://developers.weixin.qq.com/miniprogram/en/dev/reference/wxs).

Xanthopoulos, S., and Xinogalos, S. 2013. "A Comparative Analysis of Cross-Platform Development Approaches for Mobile Applications," in *Proceedings of the 6th Balkan Conference in Informatics*, BCI '13, New York, NY, USA: Association for Computing Machinery, September 19, pp. 213–220.

Zapier. 2016. "What Is Zapier?," *Zapier*, August 4. (https://zapier.com/learn/getting-started-guide/what-is-zapier/, accessed May 1, 2020).

## Copyright