Evaluation of a development framework for a mobile gaming platform with financial transfers

R. Hable[*], E. Platzer[*]
[*] Evolaris Next Level GmbH, Graz, Austria
richard.hable@evolaris.net

Abstract - Mobile gaming is a research field of growing interest due to increasing usage numbers and revenue rates. The increases are mostly caused by improvements of mobile devices and networks that enable better performance and usability of mobile games. In this paper a new framework for the development in a mobile gaming platform environment is presented. The gaming platform includes financial transfers that require specific precautions in the development process as well as in the maintenance of the platform. The development framework is evaluated in a comparative research design against other available frameworks that also meet most of these specific requirements. Evaluation criteria applied in the comparative setting are efficiency and effectiveness in form of development effort and expenses. As a result we present the differences between frameworks as well as disadvantages and advantages of our framework regarding economic as well as technical target values.

I. Introduction

We present and evaluate a framework intended to support mobile gaming on mobile phones.

The requirements for products based on this framework include supporting a large range of devices by all major mobile phone producers and operating systems. The games have to be competitive on both simple low-end mobile phones and powerful high-end smart phones. In addition to the gaming component, it is also necessary to support financial transfers based on game success. Thus, user account handling, transaction security, and prevention of fraudulent manipulation have to be an integral part of the system.

It is obvious from the requirements that tightly integrated server and client components are required to allow game development and operation with reasonable investment of resources. Existing frameworks do not meet these demands. Therefore, a new framework has been designed based on the comprehensive requirements and the current state of technology.

Major challenges during framework design were finding reasonable compromises between development effort and gaming user experience, and facilitating both responsiveness and security of game play with limited connectivity on the mobile Internet.

In the following chapters we first describe the current state of practice concerning development of client-server platforms for mobile devices. Then we present the main design decisions and structures of our framework. Finally, we show the results of a comparison with other frameworks and concepts based on the method of comparative analysis. The paper concludes with some remarks on both a general and a specific level.

II. State of Practice

A. General Situation

Contrary to server software and desktop client software, development for mobile devices is characterized by fragmentation into several largely varying operating systems and frameworks with non-neglectable market share [1]. Additionally, the distribution of market share and even the operating systems used by single manufacturers are changing rapidly [2].

B. Development Strategies for Mobile Devices

In order to provide competitive products for a large market share, normally several different versions have to be created with little, if any, code reuse. However, several frameworks are available which try to unify development on different device classes while still providing the specific look-and-feel and taking advantage of their unique hardware capabilities [3].

Another approach is to create server-centric software and thus to minimize client-specific code as much as possible, e.g. in [4]. The main task of the client software then is to interpret commands sent from the server. Most interactions with the user are based on business logic implemented by server software, which can be the same for all mobile clients.

One may even choose to create web applications instead of native applications and thus avoid installing application-specific software on the mobile devices altogether [5]. These applications rely on the common capabilities of web browsers pre-installed on modern smart phones. This, however, can lead to poor user experience caused by weak performance and the inability to use the native APIs of the mobile devices [6]. Nevertheless, platform-specific extensions can be used to approximate native look-and-feel to some extent.

C. Development of Mobile Games

The main task of many mobile applications is to provide information to the user either edited locally or gathered from the Internet. These requirements can usually be fulfilled both in native and in web applications. Mobile gaming often requires taking advantage of the full hardware and operating system capabilities of the device in order to be competitive. Thus, native programming with largely different development environments [7] can be absolutely necessary.

However, even high-performance, stand-alone games can profit from server support, e.g. in order to allow central multi-player user management and sharing of game scores [8].

In the case of mobile games which include financial transfers security aspects become additional design criteria. In principle the more operations are server-based, the more control can be enforced over user actions and thus fraud can be detected more easily.

D. *The Role of Frameworks*

When creating non-trivial applications for mobile devices, in addition to principal software design decisions it is important to carefully select tools, libraries and frameworks to be used for the implementation. Since a large number of both commercial and open-source frameworks is available, one would rarely exclusively use the tools recommended by the manufactures of mobile devices.

Both server- and client-side frameworks can be used to significantly reduce development effort and expenses. On the client side, they usually offer the opportunity to unify development [3] and thus allow code sharing across mobile device classes. On the server side, they allow implementing support services for the mobile applications based on existing software, e.g. with efficient low-level communication support [9].

III. Framework Design

A. *A New Development Framework*

The comprehensive requirements for the gaming platform necessitate coordinated selection and application of methods and tools. In particular, the necessity to support secure financial transfers requires complete control over both user actions and server transactions.

Back office support for accounting and payment can be implemented in a quite straight-forward way as server software using established technology like the Java Platform, Enterprise Edition (Java EE) [10] for security and scalability. Nevertheless, integration and deployment have to be carefully aligned with real-time game play.

Development of actual games and the required server support has to deal with a lot of uncertainties and also restrictions concerning the supported devices and game features. Therefore, this is the part where the important design decisions and concrete implementations of the new framework come into play.

B. *Deployment Architecture*

The framework has to integrate itself into the client and server landscape installed for the gaming platform. Due to a large number of potential customers using the platform at the same time and high reliability demands, the platform has to be deployed on several, partly redundant physical servers. Most of these aspects can be ignored during game development, since game play is independent from the division of tasks between the supporting servers. However, the games have to be flexible enough to connect and reconnect to different server instances according to dynamically adapted game server assignments.

The following main software artifacts are involved in the operation of the gaming platform:

- Game Client: an application running on mobile phones communicating with a dedicated game server.
- Game Server: server software supporting and controlling game play. It communicates both with an assigned number of game client instances and the back office.
- Back Office: server software supporting accounting and payment. It provides the game server with user account information and receives information about game results.

Game clients are available for different operating systems, all of them using the same communication protocol. Unified interfaces between clients and servers allow sharing code between different devices and game types. This also significantly reduces testing effort and the risk of undiscovered security holes.

The server software is tightly connected with file repositories and database management systems for game delivery and logging purposes. Reliable full-fledged logging is especially important, since financial transfers are involved, which must be reproducible particularly in case of legal disputes.

C. *A Rather Thin Client*

The most crucial decision in framework design has been to define the division of tasks between game client and game server. User experience considerations for conventional mobile games normally lead to native clients for optimal performance. Network communication with its limited speed and frequent delays is avoided whenever possible.

In our framework, however, we must maintain a tight connection between game clients and their assigned game servers. The game server always has to know about the current state of game play in order to be able to respond appropriately to situations like unexpected game termination. The user should not lose credit gained during game play due to technical problems. On the other hand, it must not be possible to cheat by interrupting game play intentionally before losing a game.

It was, therefore, necessary to find a solution which combines continuous client-server communication with high performance and responsiveness in order to gain both an attractive gaming experience and the necessary security and monitoring of game play. This was achieved in two ways: using an efficient communication protocol between the client and the server, and supporting complex independent client actions for performance-critical tasks like animated graphics. Additionally, in order to avoid having to transfer large amounts of data from the server to the client, all graphics and sounds used within a game are preinstalled.

Thus, the framework strives for a client-server model combining centralized application logic and security of thin clients with performance and user experience of rich clients.

D. *Communication Protocol*

As a consequence of the centralized application logic, communication between game client and game server is a performance critical part of the gaming platform. The framework also has to cope with widely varying hardware capabilities of the mobile devices concerning connection speed and reliability.

Based on earlier experiences with binary TCP and UDP protocols, it was decided to develop a custom binary TCP-based game message protocol (GMP) usable for all communication between game clients and servers independently of mobile device classes. This protocol supports transferring structured game data objects (GDOs), which are based on basic data types like integer values and character strings. Values are encoded as sequences of varying length, depending on the required range and precision, in order to minimize the number of transferred bytes and thus required transmission time.

GDOs are transferred within independent packets in both directions. The protocol takes care of automatic reconnection in case of connection problems and automatic recognition of transmission losses. Thus, it frees the game designer and the game client developer from handling most exceptional cases possible during communication on the Internet.

In order to prevent manipulation of transferred data all communication has to be encrypted. Therefore the TCP network connection is secured with Transport Layer Security (TLS) by default.

E. *Commands and Messages*

Client-server communication is controlled by the server sending event messages. These messages contain command GDOs which tell the game client to perform actions like drawing graphical objects on the screen or playing sound appropriate to the game flow.

Some of these commands can take a significant amount of time to perform, e.g. when showing an animation. However, the client does not have to wait for received commands to be completed before accepting new commands. Instead, it stores all commands within separate queues according to queue numbers assigned by the server. Commands within different queues are executed in parallel. In order to allow synchronized execution of commands, special compound messages are supported, which block execution within a queue until all included commands are finished.

The client may respond to server commands with information about user actions like pressing a button on the mobile phone. The server will then adapt game flow accordingly.

F. *Game Resources*

All graphics and sounds required for a game are stored in the game client. Game designers have to provide these resources for a large number of different devices. Different image formats, resolutions and color depths are used to show graphics adapted to the capabilities of these devices. Similarly, different audio formats and different bit rates are required to support sounds on all devices.

In addition to device-specific versions of resources, it is also possible to define different themes for each game. This way, different variants of the same game can be created without having to change code in the game software.

In order to easily keep track of available resources and to create device-specific game client versions, all resources of a game are specified within a device resource definition file. This is an XML file containing elements which assign unique numeric identifiers to resources referenced via file paths. These identifiers are sent to the game clients within server commands during game play for efficient retrieval of embedded resources.

G. *Code Generators*

In order to reduce development effort for changes and extensions, generators are provided which create server and client code usable by the target systems.

1) *GDO Classes*

The structure of GDO objects is defined in a custom interface definition language (IDL). An IDL file contains names and components of all data objects transferable between client and server. The GDO generator program analyzes this file and creates one class definition for each GDO. The programming language Java is used for server code and Java-based game clients. Objective-C is used for iOS clients, JavaScript for web application targets.

2) *Device Resource Classes*

In order to easily access resource objects within game clients the device resource generator program analyzes device resource definition files and creates classes retrieving resources embedded in the game client. The game client implementers can use this class to load resources like graphics and sound into memory in a format suitable for the operating system APIs.

IV. METHODOLOGY

A. *General Approach*

The framework is evaluated against other systems using the method of comparative analysis. Different criteria like development effort and user experience are examined which may even be in conflict with each other, allowing only an estimation of the quality of the achieved compromise rather than an absolute grading.

B. *Comparison Targets*

Our framework is based on several design decisions which are not always self-evident. Therefore, we do not restrict ourselves to evaluating it only against frameworks providing similar concepts and functionality. Instead, we also look at separate aspects of the framework, which could have been handled quite differently.

1) Technology

Before even considering different frameworks, technical implementations have to make basic choices about the technology used. We include technological decisions in our comparison because they are just as important as the actual features provided by different frameworks.

Sometimes such decisions are obvious: enterprise applications which should not be restricted to a single operating system will almost always use Java Enterprise Technology [10]. Server frameworks will then be restricted to products which can be used with Java application servers. Since this was the case with our framework, we do not compare it to systems based on different server technology. Instead, a comparison between the framework approach depending closely on J2EE configuration mechanisms and a similar framework with additional provisioning support [11] is made.

The requirements for our framework also include support for a large number of different mobile phone types. Therefore, we do not consider solutions based on technology which can only be used with a single class of devices. Based on this need to support mobile phones with largely varying capabilities in a uniform server environment, we compare the method chosen for network communication with other possible approaches.

2) Framework Scope

Our framework is specifically intended to support different types of mobile games with centralized server support and high security demands. Therefore, we only consider in our comparison other frameworks and methods which could be used to achieve similar goals. Stand-alone gaming frameworks or frameworks tailored for only one specific type of game are not included.

We therefore compare native client development as used in our framework with possible cross-platform development methods.

TABLE I. DEPLOYMENT ARCHITECTURES

Strategy	Consequences		
	Supported Game Clients	***Real-Time Capability***	***Development Complexity***
J2EE-based game server assignment[a]	mobile phones	no	low
Adaptable according to client types	PDAs, mobile game consoles, mobile phones, PCs, arcade machines	not on mobile phones	high

a. Framework

V. RESULTS AND DISCUSSION

A. *Deployment Architecture*

For any mobile game using centralized services e.g. to share game results, it is a logical step to distinguish between game clients and game servers. However, frameworks and possible strategies vastly differ concerning the task distribution and the server landscape used to support a potentially large number of players for different games on different platforms.

See Table I for a comparison of the main deployment options.

1) Multi-Platform Support

Reference [12] describes server support for an even wider variety of platforms than our framework: In addition to mobile phones, also potentially less connected PDAs and mobile game consoles, but also high-performance PCs, game consoles, and even arcade machines are supported. Nevertheless, the server architecture is characterized by (multi-platform) game servers and additional back-end servers for CRM and billing similar to our framework. A prototype first-person shooting game was implemented; however, only PC and arcade machine users were able to perform real-time actions, probably due to the limited network connectivity of the smaller devices.

2) Provisioning and Services

Our framework is mainly concerned with run-time game support, leaving administrative tasks to explicitly programmed and configured software on the J2EE application server level. Other platforms as described in [11] try to relieve game providers from managing the system infrastructure. Services are installed which support game provisioning, collect game metrics, and support game content distribution with a sophisticated peer-to-peer architecture for network communication.

Since in our framework all game resources are embedded within the game clients, special content distribution support is not necessary. Data transfer rates between game clients and game servers are therefore similar for all clients. Servers can be assigned fixed numbers of players per game based on empirical measurements without the need for dynamic load balancing.

B. *Communication*

With efficient client-server communication being crucial for user experience, frameworks attach great importance to highly efficient network communication when trying to allow real-time gaming even with mobile phones. Table II gives an overview of the different strategies which are described in the following subchapters.

1) Leveraging the Mobile Network

A mobile gaming platform based on the 3GPP IP Multimedia Subsystem (IMS) [13] takes advantage of the specific features of the third-generation (3G) mobile data network. Different Quality of Service (QoS) levels allow optimizing network utilization according to the needs of specific gaming tasks. The pre-defined IMS message

TABLE II. NETWORK COMMUNICATION

Strategy	**Consequences**		
	Encoding Efficiency	***CPU Load***	***Network Latencies***
Binary IP-based[a]	good	low	medium
Protocol buffers	good	low	medium
IMS Standards	medium	high	variable
Text-based (compressed)	medium (very good)	low (high)	medium (medium)

a. Framework

types can be used to support interactivities between players. The platform uses both the Media Gateway Control Protocol (Megaco) for session control and an XML-based protocol for game related information. Standardized IMS services are also used for game distribution and provide a frame for administrative tasks.

Compared to our framework, this platform offers more possibilities for fine-tuning e.g. concerning different QoS levels. Adhering to the IMS standards and protocols, however, significantly limits freedom of design of the overall architecture of the framework. Also, the principal problems of networking efficiency could not be overcome in the described prototype implementation: a game (volleyball) with high requirements concerning network delays was not playable via the GPRS network. Other measurements [14] show that the same is valid for the UMTS network.

2) *Optimizing the Protocol*

Responsiveness and throughput also depend strongly on the protocol and data format used.

When using stream-based communication such as in IP connections some kind of request-response pattern has to be implemented. In case of our framework data packets are exchanged between game servers and game clients. A simple non-standard binary encoding is used in order to minimize the amount of data that has to be transferred. According to [15] using an efficient binary format compared to uncompressed textual standard formats significantly improves performance on the smartphone. With the binary Protocol Buffers [16] format data size could be reduced by between 40 and 60 % compared to the textual XML and JSon formats. Reference [15], however, also shows that highly compressed textual formats are even more efficient than the binary protocol concerning data size.

Therefore, it seems that using a compressed textual standard format would be a good alternative to the binary encoding used in our framework. This would allow using standard libraries for encoding and decoding of messages instead of having to manually implement serialization and deserialization for the server software and the different client systems. However, it is not clear if using standard JSON or XML parsers plus effective compression libraries would be efficient enough on all supported low-end mobile phones.

3) *Coping with Network Latencies*

Even the most efficient protocol cannot eliminate the principal delays which occur during communication between game servers and game clients via mobile Internet connections. Reference [17] shows that even delays below 100 ms are noticeable by players of first-person shooter games. This shows that architectures with server-side game control are simply not suitable for some types of games.

Game designers also have to be aware that different mobile devices and reception qualities can lead to different amounts of delays. The odds of winning or losing a game must never depend on the speed of delivery of information to the user.

C. *Client Development Frameworks*

Although our framework supports game clients for a large number of mobile devices and operating systems, it does not use any existing cross-platform framework to reduce development efforts. We will compare this development approach with alternatives described in the following subchapters. See Table III for an overview of the consequences.

1) *Pure Web Applications*

Due to the requirement of efficient performance on low-end devices and competitive performance on high-end devices, offering a game client as pure web application was not considered satisfactory. However, it would be possible to aim for hybrid solutions as recommended in [6] and described in the following chapters.

2) *Embedded Web Applications*

Web applications can be embedded within native applications using tools like PhoneGap as described in [6]. JavaScript libraries like JQuery Mobile and Sencha Touch can then be used to achieve near-native look-and-feel and to access to some operating system APIs. This leads to a unified programming language (JavaScript) and unified screen design with HTML and CSS style sheets.

However, advanced JavaScript libraries are only available for high-end devices, and the framework has to provide good user experience for low-end devices too. Web applications would also lead to lower graphical performance and less efficient client-server communication on all devices due to the exclusive use of

TABLE III. CLIENT DEVELOPMENT

Strategy	**Consequences**		
	User Experience	***Flexibility***	***Development Effort***
Native Clients[a]	very good	very high	high
Pure web applications	poor	low	low
Embedded web applications	medium	medium	low
Cross-platform framework	good	high	medium

a. Framework

Web technologies.

3) Cross-Platform Frameworks

Commercial products like Rhodes Rhomobile and Appcelerator Titanium create native code for smart phones based on a unified development environment [18]. In principle, this allows creating applications with performance and user experience similar to native applications.

These products, however, are not available for simple phones. Therefore, they would only help to unify development on some smartphones targets. The additional requirement to use a different development environment and possibly a different programming language seems to make this an endeavor with doubtable benefits.

Instead of that the framework unifies development to a certain amount with common portable Java libraries.

VI. Conclusion

A plethora of different tools and frameworks is readily available for developers of both client and server software. This includes software specifically designed to support mobile gaming. Therefore, one might ask if developing yet another framework makes sense both concerning financial resource investment and successful accomplishment of project goals.

However, any major software development project requires an agreed-upon overall structure and consistent use of tooling in order to avoid creating a system resembling the tower of Babel. Overall design decisions have to be made, and generally usable support code connecting existing components for the project at hand has to be created. A framework targeted to specific project requirements is just a step further in the same direction.

The design of our framework has shown that even broad requirements like different game types on different devices can be supported largely with unified concepts. Thus, development of a gaming platform can certainly profit from this custom-made framework. Also, the comparison with other systems and alternative approaches has shown that our framework is competitive with regard to development effort and achieved product quality.

The framework has been specifically designed to allow adding new target platforms and new game types with little development cost. It is not necessary to extend existing game clients in order to allow for new game concepts. Similarly, it is not necessary to change existing games when new target platforms are added.

Nevertheless, the principal structure of the framework with its generic game clients and tight client-server integration imposes some limits on the possible types of games and achievable user experience. Network communication limits, for example, preclude real-time action games, and the necessity to support largely different target platforms prohibits game designers from taking advantage of all advanced hardware capabilities of high-end devices. Both problems will, however, become less severe in the future due to improvements in network connectivity and hardware capabilities even of low-end devices.

In summary we consider the framework a success. The design decisions proved themselves reasonable in comparison with other systems and concepts. Still there is room for improvement concerning new game types, future mobile phone platforms, and general functionality.

References

[1] D. Gavalas and D. Economou, “Development Platforms for Mobile Applications: Status and Trends,” IEEE Software, vol. 28, no. 1, pp. 77-86, Feb 2011.

[2] A. Hammershøj, A. Sapuppo, and R. Tadayoni, “Mobile Platforms : -An analysis of Mobile Operating Systems and Software development platforms,” presented at the CMI international conference on social networking and communities, Copenhagen, Denmark, 2009.

[3] S. Allen, V. Graupera, and L. Lundrigan, Pro Smartphone Cross-Platform Development: IPhone, Blackberry, Windows Mobile and Android Development and Distribution. New York, NY, USA: Apress, 2010.

[4] I. Arsov, M. Preda, and F. Preteux, “A Server-Assisted Approach for Mobile-Phone Games,” in Mobile Multimedia Processing, vol. 5960, X. Jiang, M. Y. Ma, and C. W. Chen, Eds. Berlin, Heidelberg: Springer, 2010, pp. 170-187.

[5] M. Pilgrim, HTML5: Up and Running. 1005 Gravenstein Heighway North, Sebastopol, CA 95472: O’Reilly Media, Inc., 2010.

[6] A. Charland and B. Leroux, “Mobile application development: web vs. native,” Commun. ACM, vol. 54, no. 5, pp. 49–53, May 2011.

[7] T.-M. Grønli, J. Hansen, and G. Ghinea, “Android vs Windows Mobile vs Java ME: a comparative study of mobile development environments,” Proceedings of the 3rd International Conference on PErvasive Technologies Related to Assistive Environments, New York, NY, USA, 2010, pp. 45:1–45:8.

[8] R. Hable and O. Petrovic, “A Platform for Server-Side Support of Mobile Game-Based Learning,” in Serious Games on the Move, O. Petrovic and A. Brand, Eds. Vienna: Springer, 2009, pp. 195-208.

[9] C. Xu, “A New Communication Framework for Networked Mobile Games,” Journal of Software Engineering and Applications, vol. 1, no. 1, pp. 20-25, 2008.

[10] P. J. Perrone, V. S. R. R. Chaganti, and T. Schwenk, J2EE developer’s handbook. Indianapolis, Ind.: Sams Pub., 2003.

[11] A. Shaikh, S. Sahu, M. Rosu, M. Shea, and D. Saha, “Implementation of a service platform for online games,” Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games, New York, NY, USA, 2004, pp. 106–110.

[12] J. Han, I. Kang, C. Hyun, J.-S. Woo, and Y.-I. Eom, “Multi-platform Online Game Design and Architecture,” in Human-Computer Interaction - INTERACT 2005, vol. 3585, M. F. Costabile and F. Paternò, Eds. Berlin, Heidelberg: Springer, 2005, pp. 1116-1119.

[13] A. Akkawi, S. Schaller, O. Wellnitz, and L. Wolf, “A mobile gaming platform for the IMS,” Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games, New York, NY, USA, 2004, pp. 77–84.

[14] M. Busse, B. Lamparter, M. Mauve, and W. Effelsberg, “Lightweight QoS-support for networked mobile gaming,” Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games, New York, NY, USA, 2004, pp. 85–92.

[15] B. Gil and P. Trezentos, “Impacts of data interchange formats on energy consumption and performance in smartphones,” Proceedings of the 2011 Workshop on Open Source and Design of Communication, New York, NY, USA, 2011, pp. 1–6.

[16] G. Kaur and M. M. Fuad, “An evaluation of Protocol Buffer,” Proceedings of the IEEE SoutheastCon 2010 (SoutheastCon), 2010, pp. 459-462.

[17] T. Beigbeder, R. Coughlan, C. Lusher, J. Plunkett, E. Agu, and M. Claypool, “The effects of loss and latency on user performance in unreal tournament 2003&#174;,” Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games, New York, NY, USA, 2004, pp. 144–151.

[18] T. Paananen, “Smartphone Cross-Platform Frameworks : A case study,” 2011. [Online]. Available: http://publications.theseus.fi/xmlui/handle/10024/30221. [Accessed: 30-Jan-2012].