The NX iPhone 2D Gaming Framework

Eduardo Coelho
George Ruberti Piva
Nexia Mobile Solutions

Paulo César Rodacki Gomes
Dalton Solano dos Reis
Universidade Regional de Blumenau (FURB)

Abstract

The recent release of Apple's iPhone SDK opened new possibilities for mobile game development. Currently available commercial and open-source game engines still lack support for some specific iPhone features such as NIB files, UIKit and Bonjour. This paper presents NX 2D Gaming Framework, a framework for rapid 2D game development with better integration to such specific iPhone technologies. With NX 2D Gaming Framework, developers might integrate OpenGL and UIKit interfaces in single-player and multi-player iPhone games over Wi-Fi networks using Zero Configuration Networking Standard and multi-touch interface.

Keywords:: iPhone, Mobile Game Engine, Zero-configuration

Author's Contact:

{eduardo,piva}@nexiamobile.com
{rodacki,dalton}@inf.furb.br

1 Introduction

The recent release of Apple's iPhone SDK opened new possibilities for mobile game development. iPhone has some unique features among existing mobile platforms such as its programming language, development environment, device's hardware resources, operating system and even the business model for application distribution.

This paper presents NX 2D Gaming Framework, a lightweight and compact 2D game development framework based on Apple's iPhone SDK. It's main purpose is provide developers a set of tools for rapid development of small-scale 2D game projects.

The main differences among this framework and other available engines lies in its integration with features that are exclusive to iPhone. The NX 2D Gaming Framework allows simultaneous utilization of OpenGL ES and Apple's UIKit framework. Also, it provides support for NIB files[1] and specifies a communication protocol for multi-player games over Wi-Fi networks based on Apple's Bonjour technology. In the following sections the NX 2D Gaming Framework will be referred simply as NX Framework.

Beyond this introduction, this paper is divided into 5 more sections. In section 2 some related work are presented. Section 3 shows the iPhone OS technologies, including hardware issues and operating system. Section 4 details the NX Framework architecture and its underlying technologies. For a more concrete examination and validation of results, the section 5 shows two game projects based on NX Framework. Finally, some results and future work are cited in section 6.

2 Related Work

Some related works which also aims to make the game development for the iPhone platform easier were identified, these include the open-source frameworks SIO2 [SIO2 2009], Cocos2D-iPhone [cocos2d 2009] and Oolong [Engel 2009], the commercial game engines Unity [Unity Technologies 2009], iTGB [GarageGames 2009] and Ston3D [Stonetrip 2009]. Few academic projects focusing in Apple's iPhone SDK have been published until the present date. Comparing to NX 2D Gaming Framework the cited game engines still have characteristics that are too much adherent to game development for desktop platforms.

Most of these engines provide support for iPhone specific features such as accelerometers and multi-touch screen, however they all lack support for specific technologies such as Apple's Interface Builder files, iPhone's UIKit framework and Bonjour (Apple's implementation of Zero Configuration Networking Standard). The use of those technologies brings some advantages such as better integration with the device's operating system services and user interface system, faster code development and less probability of error occurrence.

3 iPhone OS Technologies

The iPhone 3G is a smartphone powered by an ARM 620 Mhz processor with 128 MB RAM memory, 8 and 16 GB flash drive. The recent released iPhone 3G S upgrades to 16 and 32 Gb flash drive. Moreover, this device comes with proximity and ambient light sensor, multi-touch 320×480 screen, 3-axis accelerometer and magnetometer.

The iPhone OS, which is the iPhone operating system, is based on a variant of the same Darwin operating system core that is found in Mac OS X [Allen and Appelcline 2008]. Regarding the software development for this platform, Apple has released the iPhone SDK on June 2008, which includes an Objective-C compiler and IDE (Xcode), iPhone simulator, and a suite of additional tools for developing iPhone and iPod applications [Dalrymple and Knaster 2008].

The iPhone SDK provides a rich set of APIs that are usefull for game development, including multi-touch event and accelerometer support, 2D and 3D rendering with OpenGL ES 2.0, audio playing back; network communication infrastructure, data persistence, views and windows abstractions, hardware assisted animation support and so forth. The frameworks av ailable for developers are found in the iPhone OS [Apple Computer 2009a], which can be viewed as a set of abstraction layers, as depicted in figure 1.

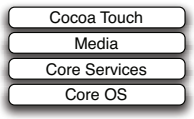

Figure 1: *iPhone OS layers*

At the lower layers of the system are the fundamental services on which all applications rely, while higher-level layers contain more sophisticated services and technologies. Higher-level layers provide object-oriented abstractions for lower-level constructs, but not necessarily mask the technologies contained in the lower layers.

The Cocoa Touch Layer comprises the UIKit and Foundation frameworks, which provide the basic tools and infrastructure necessary to implement graphical, event-driven applications in iPhone OS. The Media layer provides the key technologies to 2D and 3D drawing and audio playbing back. The most notable frameworks present on this layer are: OpenGL ES, QuartzCore, Core-Graphics, AudioToolbox, OpenAL and Media Player. The Core Services layer provides the fundamental system services that are used by all applications, such as collection data types, String date and time. Finally, the Core OS layer encompasses the kernel environment, drivers, and basic interfaces of the operating system.

[1] A NIB file describes applications user interfaces that were previously constructed in the Apple's Interface Builder WYSIWYG editor and are based on Apple's UIKit framework.

4 The NX iPhone 2D Gaming Framework

The iPhone hardware capabilities and the rich set of APIs provided by the iPhone SDK are keen on for gaming development. The NX Framework appears as an abstraction for those technologies, which makes the game development for this platform easier. By using the framework, the developer is able to take advantage of its abstraction while being able to access lower levels of iPhone OS' APIs as well.

The NX Framework is written in the Objective-C programming language and so was modeled under the object-oriented programming paradigm. The Objective-C language is quite interesting because it has a memory management system (garbage collection). Furthermore, it is compatible with C and C++ programming languages, giving to the developer a lot of flexibility concerning the technologies to be used. This language also allows better integration with the iPhone OS API than C or C++.

The NX Framework's architecture is presented in figure 2. It comprises the Engine and the Network modules. The former contains classes and protocols common to 2D game development while the latter specificaly emcompasses the network infrastructure of the framework. This architecture's proposal offers a compact set of abstractations of the Apple's frameworks (present in the lowest-level layer) that allows the quickly development of new games, by just adopting the NX Framework protocols.

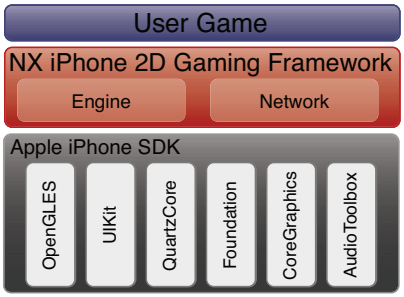

Figure 2: *NX Framework architecture*

4.1 Engine module

The **Engine** module is the game engine properly speaking, it is responsible for controlling the global functionality of a 2D game and provides classes for the game instantiation, user multi-touch event handling and game screen management. Figure 3 shows the class structure for the Engine module.

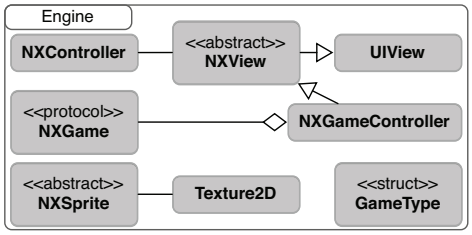

Figure 3: *Engine module class structure*

The **NXController** is the module's main class, it controls the game displaying on iPhone screen and maps the events that arises in the visualization layers to the game implementation. It also makes the transitions between classes that inherit from NXView, mananging their allocated memory. Moreover, the **NXController** class has operations that allows the presentation of game screens that can be loaded from a NIB file. Only one object of this class is instantiated and its life cycle corresponds to the whole application life cycle. Since this object has a reference to a NXNetwork class instance, it can provide to the other game classes the network communication infrastructure, if needed. At last, the **NXController** class has data persistence mechanisms, which allow game state persistence that would be recovered in a future execution.

The abstract class **NXView** represents a game view that can be displayed on the iPhone screen. It inherits all functionalities from UIView class (UIKit framework) and has, in addition, a reference to a NXController object. As a result, all its descendent classes are able to use their reference to the NXController object to request a transition between NXViews, which occurs, for instance, when the screen is changed from the game screen to the options menu screen. Interface files constructed in Apple's Interface Builder (NIB files) typically describes GUI screens. They can be loaded and rendered with the creation of NXViews sub-classes which are the NIB's "file owners".

NXGame is a protocol[2] that defines the operations that must be implemented by a 2D game. These operations comprise memory management issues and, mainly, the gameloop. This protocol also garantees that classes that adopt it have a reference to a NXGameController object, in this way, these classes are able to suspend or pause the gameloop.

The **NXGameController** is a NXView specialization for a CAEAGLLayer layer that creates an OpenGL ES context. In other words, it is a view that displays OpenGL ES content on the iPhone screen. Its main function is to instantiate the game and manage the gameloop by using control and state attributes. The actual game to be controlled is an instance of a class that adopts the NXGame protocol. Since this class specification is generic, its constructor requires a `GameType` data structure in order to know what kind of game to instantiate. The `GameType` data structure holds the game's class and whether or not it requires network communication. Finally, the abstract class **NXSprite** defines the minimal characteristics for a 2D game object, and so aggregates a Texture2D object, which is responsible for text and texture drawing. In addition, the **NXSprite** class holds the basic attributes that represents a 2D game entity. The simulation and rendering operations are defined as abstract and can be overwritten if needed.

4.2 Network module

This module provides a basic structure to create multiplayer games using client-server networking communication. It was developed using iPhone SDK classes `NSNetService` and `NSNetServiceBrowser` which adopt the zero-configurantion standard. Also, this module defines a specific communication protocol for game data exchange.

4.2.1 Zero-Configuration

Zero-configuration or Zeroconf is a network communication architecture. It's an IETF standard to manage TCP/IP networks without needed of manually configuration or a network administrator. Its goal is to let users connect their computers or devices in a local network – by Ethernet or Wireless connection – and gain access to use all available local network services. To achieve the current Zeroconf pattern, the operational system or device have to implement three functionalities: *i)* be capable to self-assign an IP address without a DHCP server (addressing); *ii)* Translate names to IP addresses without a DNS server (naming) and *iii)* discover available local network services (service discovery). Promoted by Apple Computer Inc., it is available as the Bonjour implementation [Voip-Info 2009].

4.2.2 Bonjour

Bonjour implements Zeroconf's functionalities of addressing, naming and service discovery. To addressing, the Bonjour's proposed solution is the auto-assign of IP addresses into a LAN or a network segment. The naming process is similar to the addressing one – each service or device auto-assigns a name and tests if it is already in use. Finally, the service discovery lets applications search into local network for a particular service type's instances and keep on a list of services names (which are persistent rather than non-persistent addresses), allowing a service name to be resolved in an

[2] A protocol, in Objective-C, is a list of method declarations that any class and perhaps many classes, might implement. A protocol is simply a list of method declarations, unattached to a class definition.

address and a port number always that is needed [Apple Computer 2009b].

Differing from the traditional devices oriented approach, Bonjour is service oriented. This makes requests to all devices about "what services they provide" unnecessary, just requesting about "which device provides determined service". To set a service instance name, Bonjour uses the convention: "`ServiceName._ServiceType._TransportProtocol-Name.Domain`". A valid example could be "`I-601's iPhone._airhockeygame._tcp.local.`", which represents an `airhockeygame` service type available by `TCP` connection in the `local.` domain, where the `ServiceName` is a human readable descriptive name. Each element is separated by the "_" character. Bonjour services architecture supports three basic operations: service publication, discovery and resolving.

4.2.3 Protocols

This network module of NX Framework is composed by classes presented in figure 4 and the communication protocols described below:

- NXServerDelegate: this protocol defines the operations `serverDidEnableBonjour:withName:`, `server:didNotEnableBonjour:` and `didAcceptConnectionForServer:inputStream:outputStream:` which must be implemented in a delegate class to manage a server creation and a Bonjour service publication. These operations are invoked if a Bonjour service was published, was not published or when it receives a connection, respectively;

- NXClientDelegate: defines the operations `netService:DidResolveAddress:` and `netService:DidNotResolve:` which must be implemented in a delegate class to manage a Bonjour service resolution. These operations are invoked when a Bonjour service was or was not resolved, respectively;

- NXNetworkController: defines the operation `setupNXNetwork:` which must be implemented in a class that has to control the NXNetwork object. Through this operation, the controlled object must be initialized and informed that a class which adopt this protocol is its current controller;

- NXNetworkServerController: is a NXNetwork-Controller's specialization, defining the operations `serviceDidCreate:`, `serviceDidNotCreate:` and `serviceDidAcceptConnection:` which must be implemented to control the Bonjour service publication process. These described operations are invoked by the controlled object, respectively, when a Bonjour service was or wasn't published or when a connection to a published service was established, allowing the decision making for the class that adopt this protocol;

- NXNetworkClientController: is a NXNetwork-Controller's specialization, defining the operations `serviceDidResolve:`, `serviceDidNotResolve:`, `didConnectToService:` and `didNotConnectToService:`, which must be implemented to control the Bonjour service resolution and connection processes. These operations are invoked by the controlled object, respectively, when a Bonjour service was resolved as well as when a connection to a Bonjour service previously resolved was established;

- NXStreamEventHandler: define the operation `stream:handleEvent:` that controls the stream of bytes through the input and output streams available in a NXNetwork instance. These streams are created after a connection to some Bonjour service's server.

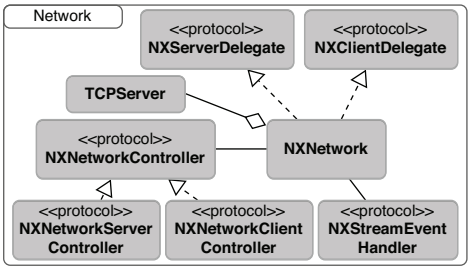

Figure 4: *Compact Network module's class diagram*

4.2.4 The NXNetwork class

Having commonly only one instance in an application, this class adopts the `NXServerDelegate` and `NXClientDelegate` protocols, allowing itself to publish, discover or resolve a Bonjour service. Once defined the domain, the service type and the transport protocol, it becomes possible to execute the three basic Bonjour operations (service publication, discovery and resolution) through the operations `publishService:` and `browseAndResolveService:`. The operations are performed as described bellow:

- publication: instantiates a TCPServer class object – responsible to create a TCP server – and publics the Bonjour service by the `publish:` operation from the `netService` attribute;

- service discovery: uses an instance of `NSNetServiceBrowser` – responsible to return all Bonjour services of a selected type – to update the `netService` attribute with the desired service;

- resolution: calls the `resolveWithTimeout:` operation from the `netService` attribute.

During the execution of the above operations, the reported events are notified to the `activeNetworkController` attribute – a reference to an object that adopts the `NXNetworkController` protocol – which should be informed through the `setActiveNetworkController:` operation. After a connection through the `connectToService:` is established, this class is responsible for opening the input and output streams – instances of `NSInputStream` and `NSOutputStream` – and resend these streams' reported events to the `activeStreamEventHandler` attribute. This attribute keeps a reference to an object that adopts the `NXStreamEventHandler` protocol, which should be informed through the `setActiveStreamEventHandler:` operation to transmit bytes over a network. Lastly, the `stop:` operation finishes the current active service.

5 Samples

At the moment two game projects were developed using the NX Framework. The first one, Zig Zig Zaa [Coelho et al. 2009] is an application that was designed for educational purpose and contains two embedded games. It is already published on Apple's App Store. The second one is an AirHockey game (named here as NX-AirHockey), which will be introduced here as a study case.

In the NXAirHockey game, matches can played by two players simultaneously in the same device (by using the multi-touch capabilities) or even on different devices (by using the NXNetwork infrastructure). A single player game mode is also supported, in this case, the player plays against the computer that is controlled by the artificial intelligence module that was implemented. The game architecture comprises two modules: Model and Game. The Model module aggregates the game domain entities, which are subclasses of NXSprite. The Game module actually do the game implementation by making the interaction among the entities defined in the Model module. It also encompasses the game menus and connection screens, along with the gameplay screen itself. The class diagram shown in figure 5 depicts the classes responsible by the game-

play implementation. The `NXController` class has the function of present to the user the game graphical content on the iPhone's screen. In this case, this class aggregates a `NXGameController` object, which is a specialization of a `NXView` for a CAEAGLLayer layer. This is due the fact the game rendering is done by OpenGL ES.

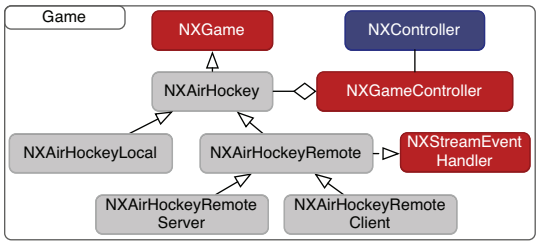

Figure 5: *NXAirHockey Game module (gameplay)*

The `NXAirHockey` class serves as the basis for the other game implementation classes. It adopts the `NXGame` protocol and therefore implements the gameloop and event handling operations. By doing this, it is capable to manage the game rules. Some specializations of this class were made in order to distinguish instances of games that occur locally or remotely.

The class `NXAirHockeyLocal` comprises the implementation of a game that occurs locally, either in single player or multiplayer mode. Regarding the implementation of a remotely game, after established a client-server connection between two game instances running on different devices, confirmation messages are sent through the input and output opened streams, in order to know the moment the game can start. Afterwards, a `NXAirHockeyRemote` descendant class object is instantiated, a `NXAirHockeyRemoteServer` instance in the case of a game server instance or a `NXAirHockeyRemoteClient` in the case of a client game instance.

The `NXController` class is capable to present OpenGL based views as well as views composed by UIKit components (NIB files) into the iPhone's screen. Thus, menus and screens similar to a connection screen were generated in the Interface Builder, hence being easily and quickly developed and edited. This approach is particularly desirable because UIKit components are more functional for user interface such as menus and configurations screens than Open GL. The figure 7 presents the `NXController` showing the two different screen types. View transitions are smoothed through Core Animation (QuartzCore framework) features.

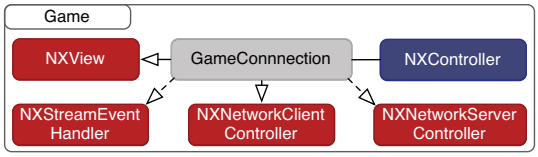

Figure 6: *NXAirHockey's game module*

The figure 6's diagram shows the `GameConnection` class being presented into iPhone's screen by the `NXController` (figure 7) and also presents the class managing connection between different game instances, through the protocols `NXNetworkServerController`, `NXNetworkClientController` and `NXStreamEventHandler`. In addition, the class is responsible for publishing or resolving a Bonjour service, respectively, when running in server or client mode.

6 Conclusions

This paper discussed the development of the NX 2D Gaming Framework, which brings an alternative option for the development of smaller scale game projects. Comparing to the development based purely in iPhone's SDK APIs, our framework offers some advantages such as a network communication abstracion layer based

Figure 7: *(a) Game connection screen (loaded from a NIB file); (b) NXAirHockey game screen (rendered into an OpenGL canvas)*

on Zeroconf standard, integration with user interfaces constructed with Apple's Interface Builder and integration with Open GL ES. The use of NIB files (from Interface Builder) for non gameplay classes leads to a faster development of menu and others views. This advantage was not found in the related works. Future improvements include game interface support based on iPhone's accelerometers and support for audio and music, which is currently made by third party libraries.

Acknowledgements

The authors would like to thank University of Blumenau (FURB) and Project Acredito for the finantial support for equipment aquisition.

References

ALLEN, C., AND APPELCLINE, S. 2008. *iPhone in Action: Introduction to Web and SDK Development*. Manning Publications Co., Greenwich, CT, USA.

APPLE COMPUTER, 2009a. iPhone OS technologies. `http://developer.apple.com`, May.

APPLE COMPUTER, 2009b. Bonjour overview. `http://developer.apple.com`, April.

COCOS2D, 2009. cocos2d for iphone. `http://www.cocos2d-iphone.org/`, June.

COELHO, E., PIVA, G. R., AND GOMES, P. C. R., 2009. ZigZigZaa - Apple App Store. `http://itunes.apple.com/WebObjects/MZStore.woa/wa/viewSoftware?id=306502103&mt=8`, Mar.

DALRYMPLE, M., AND KNASTER, S. 2008. *Learn ObjectiveC on the Mac*. Apress, Berkely, CA, USA.

ENGEL, W., 2009. Oolong engine. `http://oolongengine.com/`, June.

GARAGEGAMES, 2009. iTGB. `http://www.garagegames.com/products/torque-2d/iphone`, June.

SIO2, 2009. Home. `http://sio2interactive.com/`, June.

STONETRIP, 2009. Stonetrip: Development is a game. `http://www.stonetrip.com/`, June.

UNITY TECHNOLOGIES, 2009. Unity: Game development tool. `http://unity3d.com/unity/`, June.

VOIP-INFO, 2009. Asterisk zeroconf support - voip-info.org. `http://www.voip-info.org/wiki/view/Asterisk+Zeroconf+Support`, June.