Breder Programming Language for iPhone Game Programming

Bernardo Breder
Universidade Federal Fluminense

Esteban Walter Gonzalez Clua
Universidade Federal Fluminense, Medialab

Abstract

This paper, propose a new framework definition for programming to iPhone with simplicity and productively. With this way, the developer can create a application, thinking only the logic of the game, forgeting the difficulties of framework and native Language for programming to iPhone.

This framework has simple and objective features to ensure the practicality of programming for iPhone. For this, we implemented a specification based on typical game framework architectures, which meets these requirements by using a Programming Language.

The Objetive-C Language used for programming to iPhone is a low level Language[STEPHEN G. KOCHAN]. With this, the developer need to implement and manager many thinks in the environment that should not worry. Because of that, it will be good if the developer can use a high level Language in which will only worry about the logic of the application.

In this paper, will be used the Breder Programming Language for implement this framework. It because, the Breder Language is a high level Language that has many features to ensure the further structuring of code and productivity [BREDER, B]. Moreover, this Language was developed to assist projects that require high efficiency in native method call deployed in some external library, that will be use for comunication with iPhone framework [BREDER, B].

Keywords:: Breder Programming Language, Breder Virtual Machine, Garbage Collector, iPhone, XNA, Framework

Author's Contact:

bernardo@breder.org
esteban@ic.uff.br

1 Introduction

The superiority in graphics applications with iPhone, iPad, a significant increase in developing programs for this platform overcame in the latest years. With the Apple's application called App Store, facilitated the distribution of commercial software developed for the iPhone, either for big and small developers.

New applications increasingly challenging the limits of the hardware. In addition, several other ways to develop an application for the iPhone are being used to meet the needs of a better development.

Currently, many applications have been developed so unproductive. This is happened because to develop an application, you must use a native Language of iPhone called Objective-C which not provide many features for the developer. Moreover, the standard library of the iPhone has many mechanisms that enable the occurrence of failures due to their misuse. It because the framework of iPhone is a low level and the developer need to study so much for programming to iPhone. With this, if you call a operational in wrong context, at runtime, the process can be killed. To make a development of a game in Objective-C Language, it is necessary to worry about several features of the Language and game frameworks typically offer.

Moreover, the Objetive-C Language is not a easy for programming. This because in this Language, the developer need to worry about many thinks, that he should not worry. For example, if the context is to paint a frame, the developer must use only operation about paint. But, the Objetive-C Language and the Framework permit to use other operation what can't use in this time. It because the framework use the state machine to work, permiting to use operation in wrong time. With a Language in the front, with the organization of the class, the Language can previne this operation.

A framework is responsible for implementing a specific functionality. Furthermore, the framework should have mechanisms that allow the configuration of its execution. Unlike libraries, the framework dictates the flow control application, called Inversion of Control [Martin Fowler, 2004]. Inversion of Control is an abstract principle describing an aspect of some software architecture designs in which the flow of control of a system is inverted in comparison to procedural programming. In traditional programming the flow is controlled by a central piece of code with callbacks. Using Inversion of Control this central control as a design principle is left behind.

For example, the framework will ask how to paint the player in the game or what happend if i touch the screen in a part of the game. But, the standards frameworks provided by Apple to develop applications for iPhone, are very sensitive to failures, in which the framework will try to resolve conflicts in a better way.

The aim of this paper is to establish a environment of development for create game with simply, secure and productive way, minimizing the failures and the complexity of this the iPhone environment. For this, we created several abstractions on top of a framework, both to meet the difficulties of standard frameworks, how to address the limitations of the Programming Language Objective-C.

The great advantage of the Breder Language is the fact that it had a very efficient Virtual Machine for native operations. Thus, the framework will use this feature to be implemented more efficient yours methods. In addition, this Virtual Machine is concerned with embedded environment. This way, different techniques will be adopted so as not to harm the functioning due to lack of resources.

In the next section will be presented limitations and unproductive in the development of an application for the iPhone and will also be discussed solutions to such problems. Session 3 will present the definition of the Breder Language, specifically created for this work. In session 4 we will present the specification of a framework able to address such described limitations and completelly based on the Breder Language. Session 5 describes how to implement a framework using our proposed Language. In session 6 we will present a demo built on our proposed solution. Session 7 presents considerations necessary to maintain this environment with performance. Finally, the last session presents the conclusions and future issues of our proposal.

2 Limitations in programming for the iPhone

Programming for Objective-C Language does not provide all the resources necessary for a good development. It because it is very low level programming, where the developer need to know and manager many thinks for programming to iPhone SDK. For the developers, they will try to find a new development environment for programming simplely, investing you time in the logic of the aplication and not with thinks that the Language need to work. Would be good if the development environment uses high level popular Programming Language.

With this in mind, it should be established the usage of a more popular Programming Language with better features, such as Java, C++, Python, among others. But, this Language need be a high level, because the programmer do not want to worry with litter thinks, only with the logic of the aplication.

Moreover, Objective-C Language has no abstract capabilities enough to meet an easy implementation of complex features, typically required in games. This is because the Language does not provide tools, abstract features and libraries that facilitate the implementation, making it necessary to develop such resources.

For example, the Objective-C Language does not provide a complete memory management transparently [STEPHEN G. KOCHAN]. In this way, the developer can use unauthorized memory in a wrong way, causing lower permanence manipulation of objects in memory or even the fall of the application execution. When this occurs, it is unproductive the detection and clearance of the problem.

To resolve the lack of transparency in the management of memory, it Is necessary to create mechanisms to release the objects automatically. That would be the mechanism to recognize the objects that are no longer being used, releasing them from the memory. To implement such a mechanism, it is necessary to implement the concept of Garbage Collection in a transparent manner.

Another thinks that Objective-C Language is not so good for programming to iPhone, is because do not have a safe executing. It because, the application is compiled for the machine and if a error happen, the process will be killed. It will be good is the executing of the application is manager by a Virtual Machine. With this way, a error will not kill the process, doing the execute more safe.

Moreover, the best way to capture erros is in the compiler timer. If you have a error in runtime, this is a unproductive development to right this error. At compiler timer, the error is better and easier to resolver be, because the errors are not provided in a specific context.

If the error is in runtime, the developer need to execute the application for understand what happening. With this, the developer need to know what context that is happening and why. Thus, a good structure of language to retrieval errors at compile time, can generate greater productivity.

Apple offers several complex framework and low level development specifically for the iPhone platform. However, its misuse can result in failures that are difficult to deal. To resolve the complexity of the standards framework, the developr must create an abstraction in order to reduce failures that may occur. This will create operations that do not generate side effects if they are called improperly.

3 Breder Programming Language

In order to solve problems related to the Language Objective-C, such as the lack of resources, we propose and implement in this paper a definition of a new high level language. This Language is focused on structuring and organizing the source code in order to increase productivity and safety in the implementation of an application. Also, this Language have a syntax similar to the Java Language in order to make its learning more practical and follows completely the Object Oriented paradigm.

The objective of the proposed language is providing tools for developers to obtain a clear code, simple and with a high level standard. Breder Language is strongly typed, allowing various syntactic and semantic errors detections at compile time, facilitating the development. Although the language is very useful for iPhone developers, it can also be used in many other pourposes.

Breder Language was developed for Windows, MacOS and Linux, allowing the use of the same source code in other environments. Therefore, we developed a Breder Virtual Machine that abstracts all the features of specific hardware and Operating System, thus maintaining homogeneity in the application execution and capable to be used in diverse hardware as embedded devices. In addition, several embedded devices like iPhone and Android were tested in order to calculate the degree of cost generated by Breder Language.

Breder Language has interesting features that make game development more productive and reliable. One of the main characteristics is the high Object Oriented paradigm and high level of Garbage Collection, with automatic object management in the memory.

The Breder Language has the feature of a class with type Interface [BREDER, B]. Thus, this class called Interface will not implement the methods declared. This because your goal is to keep the specification of the class that implement it. Thus, new implementation of that concept, should follow the specification of the interface, making it easier to use the class implemented, because the user already know the signatures of the methods described in the interface.

Moreover, the Breder Language has a better encapsulation of classes, protecting their fields and methods outside of classes, thus making the code more secure. Also, this Language has multiple hierarchy [BREDER, B], in which a class can extend multiple classes and interfaces. Thus, an organization and reuse of code will be better assigned, thereby facilitating the organization of classes and interfaces.

An important requirement of Breder Language is to serve projects that require intensive processing, such as games. For this, we created several internal aspects that ensure high performance in communication between Breder Language and other Compiled Languages.

The Breder Language allows the creation of native methods that were used to implement operations in a Compiled Language. We must therefore develop methods to implement such operations that need to access specific features of the iPhone SDK.

Finally, the great advantage of the Breder Language is the fact that it had a very efficient Breder Virtual Machine for native operations [BREDER, B]. Thus, the framework will use this feature to be implemented more efficiently. In addition, the Breder Virtual Machine is concerned with embedded environment. This way, different techniques will be adopted so as not to harm the functioning due to lack of resources. For example, the Garbage Collector of Breder Virtual Machine, in embedded devices, will work lightly, not hurting application performance.

4 Specification of a Framework

As mentioned previously, most problems related to programming for the iPhone can be solved with a specification of an appropriate framework that protects most of the possible failures that may occur. That should set up a high-level structure in order to isolate the internal complexities from a basic library of the iPhone SDK.

The first basic structure would be the high-level abstract representation of a window. With this, all models of iPhone will always have a single window with different sizes depending on the device, according to its resolution. For example, the window for the iPhone 4 have a dimension of 960x640 for the iPhone 3G would have a dimension of 480x320 and to iPad have a dimension of 1024x768.

Another important basic structure for a game are Components. A Component represent a object in the device, visible or not, upgradable or not and suitable for events. This component represent the most basic abstraction of a feature in the window. Thus, in a game application, a window would have at least one component to be displayed, even though this is just a static image without any events. Therefore, as the component is a structure with the possibility to be graphics, it must have a dimension of size and other typical graphics properties. The component must have a event handler, which can be modeled as a callback.

This object may change of state in the context of the application. Thus, all components registered in a window will always have a status update event at each screen update. After all the components change states, they will suffer the event painting in the entire hierarchy of components of the window.

At the same time we define a component as the most basic representation structure of an object. For this specification the framework defines a structure that represents a group of component. With this, we can define a hierarchy of components with parent-child relationship, in which all components may or may not have a father and the whole structure of group can have several components sons. This group structure will be called Container, which will be the father of several children components. The same occurs with the component. The container will also have a dimension in which a reference

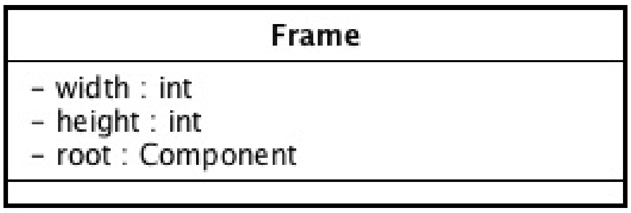

Figure 1: *A Frame class*

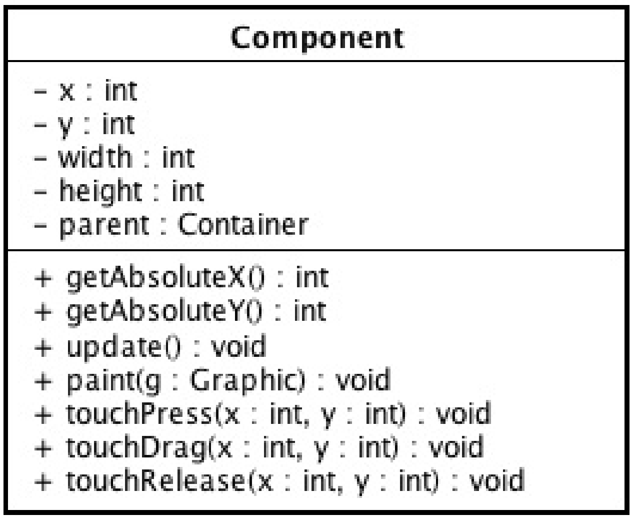
Component
- x : int
- y : int
- width : int
- height : int
- parent : Container
+ getAbsoluteX() : int
+ getAbsoluteY() : int
+ update() : void
+ paint(g : Graphic) : void
+ touchPress(x : int, y : int) : void
+ touchDrag(x : int, y : int) : void
+ touchRelease(x : int, y : int) : void

Figure 2: *A Component class*

for some operations will be defined. In the specific case of the container, when any event is released, it will always be propagated to the children, passing on their information. Therefore, every event that occurs at the Container will always be handled by some component.

In the case of a container, the state update corresponds to updating the states of all components sons. The same is true when calling its painting callback, which will be delegated to all components in the order that was registered, where the first registered will be the first to be painted.

As the Component and Container has some features in common, a class hierarchy can be defined to take advantage of features of size, event, updating and painting. So in the class diagram, a Container will inherit from a component.

As mentioned previously, a Container can have multiple children. Therefore, we must define that a Component is always a parent, unless it is the root of the hierarchy.

As shown, all components will always suffer first the action of the state update and then will be painted, in case of graphical items, at each screen update. With this, the framework creates an entity that abstracts the process of painting and the updater.

Therefore, independent of the framework of iPhone available, a new framework that abstracts typical difficulties will be created while maintaining the simplicity in its usage. For a game, basically, we defined a structure that works with primitive operations with 2D images, also called sprites.

A class organization that best represents the proposed architecture will be presented below.

Figure 1 shows a class defined for the window representation.

Class that represents a component in the figure 2, where contains the fields coordinated relative size, kinship and sensitive to the touch event. In addition, the component will have functions to retrieve absolute coordinate, change of state, painting and event handling of touch, since the act of pressing, drag and release your finger from the screen

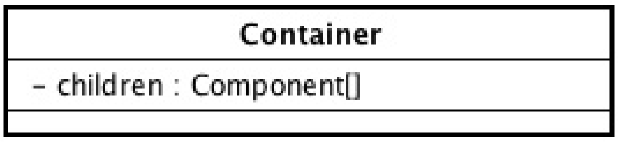

Figure 3: *A Container class*

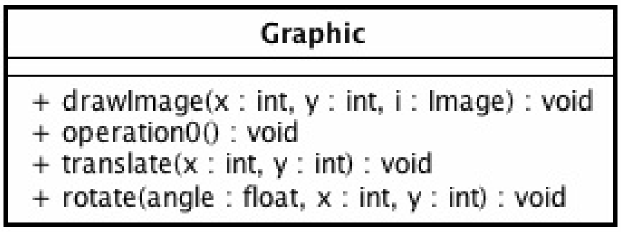
Graphic
+ drawImage(x : int, y : int, i : Image) : void
+ operation0() : void
+ translate(x : int, y : int) : void
+ rotate(angle : float, x : int, y : int) : void

Figure 4: *Example of image*

The class that represents a container is illustrated in figure 3. This container may handle the fields of Its children.

The class that represents the painting, which contains primitive functions to paint , rotate and move an image is Illustrated in the figure 4.

The class that represents an image from a file defined in the class constructor is shown in the figure 5.

Since in Breder Language is possible to detect, at compile time, if a parameter or a return of a method may be a possibly null value or not[BREDER, B], the implementation of the framework will use this feature to tie all the parameters of the methods, telling what parameters of the methods will be null or not, causing them to become more productive and reliable to use.

The native methods are operations implemented in a Compiled Language. In this work these methods are implemented in ANSI C. This is an important strategy in order to guarantee high performance and for , decreasing the overhead in the Breder Virtual Machine processing. For the sake of performance, most of the methods of the game framework will be implemented natively.

5 Results

In order to validate our language definition and the framework built upon the language, we will present some interactive demos that will be executed at the iPhone environment. Based on these tests, we will determine the overheads of processing the Breder Language.

The demo created for this paper, builds an environment full of game objects, composed by sprites, and making different movements. As the purpose of the work is not the development of a full game, each game object make simple animations, which could be substituted by more complex behaviors.

As mentioned previously, the order of the children of a Container indicates the order in which objects are painted. Moreover, the touch event also follows the same criteria. So, when a touch event occurs, the components that are painted on the front will be privileged in the searching process.

In this example, the sprites are in a privileged ordered in relation

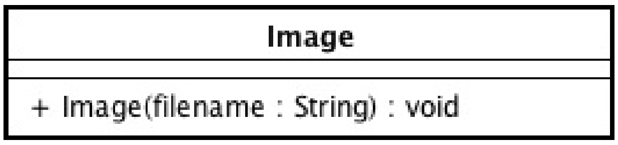

Figure 5: *A Image class*

to the touch events. As stated above, the privilege is in the order in which the components were added in the Container. The last Component added will be the first listener first. This shows that it is possible to build a hierarchy of event handlers for the game objects.

Similarly what happens to the balls, also suffer Sprites events update state. From this event, you can change how the cells of sprites were chosen as the execution context of the application.

6 Performance analysis

In this session, we will comment discuss and analyze the performance obtained using our language and framework. on the assumptions that were necessary for an Interpreted Language can process so tolerable by applications. As mentioned previously, in order to.

Several optimizations are were designed to meet the processing overhead that leads to the Breder Virtual Machine, such as the development of native methods. . For example, if the framework uses native methods, will decrease so much that overhead.

Besides the optimization of native methods that has been widely used both in the framework, as in the standard library available in Language, Breder Virtual Machine was built with the purpose of meeting and seeking the execution of native methods. Thus, when there is a need to perform native operation, the Breder Virtual Machine will not generate any kind of processing overhead.

The Garbage Collector is designed with goal of releasing the unused objects from memory. Beyond this, the Garbage Collector finds the projects of great real-time processing. In the case of applications for iPhone, as the Garbage Collector is acting in the release objects from memory the image can not be frozen. In the demo, did not find any kind of freezing as a function of performance of the Garbage Collector.

The example set by the demo, shows the degree of stress graph that was generated to test if the programming environment does not fit in projects with many images. In this demo, we printed more than 700 components at the same time, in which each exchange screen, will always be updated on her condition and repainted. Even with the polluted environment of images, the result was very satisfactory, since that was not found any sign of slowdown in the application.

The same is true when some event has touch screen. The framework will cover all the components looking for the children who suffer the event. This search has become very fast, as it was performed several optimizations on the query.

7 Conclusion

In this paper we proposed a new language architecture, a game framework based on it and a simple application to validate. This applications where developed for an iPhone platform and showed the possibility to create games without the usage of Objective C. Our work also validate the possibility to develop games for iPhone without the need of knowing its specific architecture, thanks to the Breder Virtual Machine.

In addition, several optimizations were made with the framework, in order to improve its funtionality. The frame rate obtained for a high number of game objects validates the framework for a high level game development.

The suggested development environment of an application for the iPhone, using the Breder Language, would be the Eclipse and XCode. Eclipse became simpler to use the framework since the Compiler acts all the time. So, when change is made in the source code, automatically, the Compiler will recompile the source code. This feature makes the usage of Breder language very simple to make any modification in the source code, making very productive development.

Acknowledgements

The authros would like to thank for review the content of the paper. Thanks the users that test the application builded with Breder Language.

References

BREDER, B., 2010. Breder Language Documentation. Avaliable at: *http://www.breder.org/doc*.

BREDER B., 2010. Breder Language API. Avaliable at: *http://www.breder.org/api*.

JOSELLI, MARK ; ZAMITH, MARCELO ; CLUA, ESTEBAN ; MONTENEGRO, ANSELMO ; LEAL-TOLEDO, REGINA ; CONCI, AURA ; PAGLIOSA, PAULO ; VALENTE, LUIS ; FEIJ, BRUNO ; CLUA, E. W. G. . An adaptative game loop architecture with automatic distribution of tasks between CPU and GPU. Computers in Entertainment : CIE, v. 7, p. 1, 2009.

CLUA, E. W. G. . Arquitetura e Processos de Desenvolvimento de jogos 3D de Terceira Gerao. In: XXI Simpsio Brasileiro de Engenharia de Software, 2007, joo Pessoa. Mini-cursos do XXI Simpsio Brasileiro de Engenharia de Software. Porto Alegre : Sociedade Brasileira da Computao, 2007.

JOSELLI, M. ; CLUA, E. W. G. ; MONTENEGRO, A. ; CONCI, A. ; PAGLIOSA, P. . A new Physics Engine with Automatic Process Distribution between CPU-GPU. In: Sandbox 2008: An ACM Siggraph Videogame Symposium, 2008, Los Angeles. Proceedings of Sandbox 2008: An ACM Siggraph Videogame Symposium. Los Angeles : Association for Computing Machinery, 2008. v. 2. p. 149-156.

PAUL ZIRKLE AND JOE HOGUE, 2010. iPhone Game Development, 2010.

MARCELO COHEN AND ISABEL HARB MANSSOUR, 2006. OpenGL, uma abordagem prtica e objetiva, 2006.

RICHARD S. WRIGHT, JR., BENJAMIN LIPCHAK AND NICHOLAS HAEMEL, 2007. OpenGL, SuperBible, Fourth Edition, Comprehensive Tutorial and Reference, 2007.

ALLEN SHERROD, 2007. Data Structures and Algorithms for Game Developers, 2007.

JOACHIM BONDO, DYLAN BRUZENAK, STEVE FINKELSTEIN, OWEN GOSS, TOM HARRINGTON, PETER HONEDER, FLORIAN PFLUG, RAY KIDDY, NOEL LLOPIS, JOE PEZZILLO, JONATHAN SAGGAU, BEN BRITTEN SMITH, 2009. iPhone Advanced Projects, 2009.

Apple Develper, 2010. Apple Develper Library. Avaliable at: *http://developer.apple.com/iphone/index.action*.

DAVE MARK, JEFF LAMARCHE, 2009. Dominando o Desenvolvimento no iPhone. Explorando o SDK do iPhone, 2009.

STEPHEN G. KOCHAN, 2009. Programming in Objective-C 2.0. A complete introduction to the Objetive-C language for Mac OS X and iPhone development. Developer's Library, 2009.