Breder Programming Language for iPhone Game Programming

Bernardo Breder and Esteban Walter Gonzalez Clua

Universidade Federal Fluminense
bernardo@breder.org
esteban@ic.uff.br

http://www.breder.org

Abstract. This paper proposes a new framework definition that makes programming for iPhone simple and productive. In this paper the Breder Programming Language will be used to implement this framework. That is because the Breder Language is a high level Language that has many features which ensure further productivity and structuring of code.

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

1 Introduction

Many applications have been developed very unproductive. This has happened because when developing an application you must use a native Language of iPhone called Objective-C which does not provide many features for the developer. Moreover the standard library of iPhone has many mechanisms that enable the occurrence of failures due to their misuse. That is because the iPhone framework is low level and the developer need to study a lot to program for the device. So, if you call an operation in a wrong context at run-time the process can be killed. To develop a game in Objective-C Language it is necessary to worry about several features which the Language and game frameworks typically offer.

Moreover, the Objective-C Language is not easy to program with. In this Language the developer needs to worry about many things that he should not be worrying about. For example: if the context is about painting a frame, the developer must use only paint operations. But the Objetive-C Language and the Framework permits the use of other operations which can't be used at this time. That happens because the framework uses the state machine to work, permitting the use of operations at the wrong time. The Language will allow the developer to execute certain operations only in the right context, turning the execution of operations in the wrong context impossible.

The great advantage of the Breder Language is the great efficiency of the Virtual Machine for native operations. Thus the framework will use this feature to implemenet more efficiently yours methods. In addition, this Virtual Machine is concerned with an embedded environment. This way different techniques will be adopted so as not to harm the functioning due to lack of resources.

2 Limitations in programming for the iPhone

The Objective-C Language does not provide all the resources necessary for a good development. It is a very low level language, where the developer needs to know and manage many things to program for iPhone SDK. Developers will try to find a new development environment that makes programming simpler, investing their time in the logic of the application and not with things that the Language should handle. It would be good if the development environment used a high level popular Programming Language.

Moreover, if the developer wants to implement a complex feature, typically required in games, the Objective-C Language does not have high level features enough to make this implementation easier. This is because the Language does not provide tools, abstract features and libraries that facilitate the implementation.

For example, the Objective-C Language does not transparently provide a complete memory management [STEPHEN G. KOCHAN]. This way the developer can use unauthorized memory in a wrong way. When this occurs, the detection and clearance of the problem is unproductive.

To resolve the lack of transparency in the memory management, it is necessary to create mechanisms that 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 thing that makes Objective-C Language not so good for programming to iPhone, is lack of a safe execution. The application is compiled for the machine and if an error occurs, the process will be killed. It is better if the execution of the application is managed by a Virtual Machine. This way, an error will not kill the process, showing the stack trace, making find and fix the code easy.

Moreover, the best way to capture errors is in the compiler timer. If you have an error in run time, it is unproductive to right this error. It occurred in a specific run time context, thus you need to spend time to reproduce this context to fix the problem. At compiler timer the error is better and easier to be resolved, not depending of the context of run time. So it will be good if the Programming Language use the compile time to find almost all errors as possible.

3 Breder Programming Language

In order to solve the problems related to the Objective-C Language, such as the lack of resources, we propose 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 of 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 with 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 for many other purposes.

Breder Language was developed for Windows, Mac OS and Linux, allowing the use of the same source code in many environments. We developed a Breder Virtual Machine that abstracts all the features of specific hardwares and Operating Systems, maintaining homogeneity in the application execution and allowing its use in many devices. In addition, several embedded devices like iPhone and Android were tested in order to calculate the cost degree generated by Breder Language.

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

Moreover, the Breder Language has a better class encapsulation, 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 the communication between the Breder Language and other Compiled Languages. With this, the developer can choose where to implement part of the project. If this part has much processing, the developer can program with C Ansi Language, for example. But if this part of the project has many complex features, the developer can programm with high level Breder Language.

Finally, the great advantage of the Breder Language is the great efficiency in 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 the application performance.

4 Specification of a Framework

As mentioned previously, most problems related to programming for iPhone can be solved with the 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 has a dimension of 960x640, for the iPhone 3G would have a dimension of 480x320 and for iPad, 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 it is just a static image without any events. Therefore, as the component is a structure with the possibility to be graphic, 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 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 state, they will suffer the painting event in the entire hierarchy of components registered in 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 components. This way 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 the group can have several son components. 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 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 they were registered, where the first registered will be the first to be painted.

As shown, all components will always suffer the action of the state update first 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 available iPhone framework, 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.

5 Results

In this session, we will comment, discuss and analyze the performance obtained using our Language and framework.

Several optimizations 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 many native methods, it will provide a low execution overhead.

From the native methods optimization, widely used both in the framework and in the standard library available in the Breder Language, the Breder Virtual Machine will enjoy the most of the performance, not generating any overhead. Thus, when there is the need to perform native operations, the Breder Virtual Machine will not generate any kind of processing overhead.

The Garbage Collector is designed to release the unused objects from memory. In the case of Real-Time applications, the Garbage Collector releases unused objects in order not to mess with the repainting of the screen. In the demo, it was not found any type of freezing due to the Garbage Collector.

In order to validate our Language definition and the framework built upon the Language, we will present some interactive demos that will be executed in 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 makes simple animations which could be substituted by more complex behaviors.

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, more than 800 components were printed at the same time. Each screen exchange updates the components state. Even with the image polluted environment the result was very satisfactory, since no sign of slowdown was found in the application.

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

6 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 in the framework, in order to improve its functionality. 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 iPhone using the Breder Language would be the Eclipse and XCode. Eclipse makes the use of the framework simpler since the Compiler acts all the time. So, when a change is made in the source code the Compiler will automatically recompile it. This feature makes any modification in the source code of Breder language very simple, turning the development very productive.

References

1. BREDER, B., 2010. Breder Language Documentation. Avaliable at: *http://www.breder.org/doc*.
2. BREDER, B., 2010. Breder Language API. Avaliable at: *http://www.breder.org/api*.
3. 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.
4. 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.
5. 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.
6. 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.
7. PAUL ZIRKLE AND JOE HOGUE, 2010. iPhone Game Development, 2010.
8. RICHARD S. WRIGHT, JR., BENJAMIN LIPCHAK AND NICHOLAS HAEMEL, 2007. OpenGL, SuperBible, Fourth Edition, Comprehensive Tutorial and Reference, 2007.
9. ALLEN SHERROD, 2007. Data Structures and Algorithms for Game Developers, 2007.
10. DAVE MARK, JEFF LAMARCHE, 2009. Dominando o Desenvolvimento no iPhone. Explorando o SDK do iPhone, 2009.