A GENERIC ARCHITECTURE FOR MULTI-PLATFORM WIRELESS GAME DEVELOPMENT

Alexandre Damasceno
Börje Karlsson
Danielle Rousy D. da Silva
Informatics Center (CIn)
Pernambuco Federal University (UFPE)
Av. Prof. Luiz Freire, s/n, CIn/CCEN/UFPE, Cidade Universitária,
Recife, Pernambuco, Brazil
50740-540
E-mail: {algd,bffk,drds}@cin.ufpe.br

KEYWORDS
Wireless games, game architecture, game engine.

ABSTRACT

With the current growth of the wireless devices (especially cell phones) and digital games markets, several manufactures started providing some infra-structure for the development of games for their products. With the creation of a myriad of technologies, developers realized the need of some sort of multi-platform architecture that could help game development across devices and technologies. Considering the above and the authors experience on the development of several games, this work presents a simple, generic architecture, easily adaptable to help the wireless game developer in developing mobile games quickly and efficiently a across devices such as cell phones.

INTRODUCTION

The digital games market is a market going through a huge growth, moving annually millions of dollars worldwide (IDSA 2003). In turn, the wireless devices market, is also facing a high growth, as cell phones stop being devices dedicated to voice calls and start to incorporate features like calendars, e-mail readers, MP3 players. It is not a surprise that such devices are being explored by the electronic entertainment industry as platforms for digital games distribution.

Initially, the device manufactures were the game producers, creating applications and games specifically for their handsets. But as demand for new applications and games grew, the industry realized that there was a need for the creation of some way to allow other companies to develop these applications. Pushed by this need, several device manufacturers and software companies decided to design software and hardware layers to fit between the devices operational systems and applications, allowing other companies and developers to work with this layer. One of the first initiatives in this direction was Sun’s Java 2 Micro Edition (J2ME), a “compact” version of the Java language and platform focused on devices with smaller processing power and little memory. With this support, device manufacturers could implement a Java virtual machine on their devices to run Java byte code. The adoption of such solution allowed the development of a much higher number of applications in much less time and combined with the availability of Java developers that knew the standard Java platform.

Some of the greatest difficulties on developing mobile games are the restriction imposed by the devices and the wide range of available development technologies. The situation gets even more complicated when the application domain is a complex one, as is the case with digital games. Also, many of the techniques and architectures applied in computer game development can not be directly applied to cell phone games, because these techniques usually require high processing loads and have a large memory. Considering these aspects, this work tries to present a generic architecture that is both simple and easily adaptable to help a quicker and more efficient game development across devices and technologies, but is primarily focused on cell phone games.

This architecture was created based on the experience acquired during the development of several games (Arruda 2002) by the team of which the authors are part and was used on various games as is the case of GoldHunter, SpaceRunner, PodRace, Atlantis etc. (see Figure 1).

Figure 1 – Stalingrado and PodRace

PROPOSED ARCHITECTURE

The proposed architecture was based on the experience of a Research & Development team of which the authors are part and that has already produced several development frameworks (Pessoa 2001; Barros 2003; Nascimento 2003), extensions (Karlsson and Ramalho 2002) and more than 20 games for various wireless platforms; and allowed for faster game development and good code reuse.
It is divided into five main layers: the presentation layer; the control layer; the game logic layer; a canvas layer; and an auxiliary layer. They are related as shown in Figure 2.

- The presentation layer is composed of the classes responsible for the game presentation screens, game menus (setup menus, main menu etc.), animations, and every screen necessary for the navigational model before the player gets to the game itself.
- The control layer is the game controller, responsible for the screen changes in the presentation layer. Depending on the user input, this layer will provide the necessary support to change the game screens and the game state. When a new game is started, the controller needs to launch a parallel process to handle the game life cycle until the game is finalized. This looping cycle is responsible for dealing with the game logic layer, updating its information, according to its previous state and the user inputs. After that, this process must start the screen rendering in order for the user to perceive the game environment changes.
- The game logic layer has the responsibility of representing logically the game objects, game items, the player’s avatar, the opponents and allies, obstacles as well as the object attributes and the game world representation. This layer is composed basically by a component with two functions: update and key handle. The key handle is responsible for the game input mechanism sent by the game canvas. These events are handled and the game state is updated by them. The update is responsible for the world updating process, updating all the necessary game components. Each game component must implement a update method/function, that will be called with the relevant information for the object updating.
- The game canvas has two main functions/responsibilities. One of them is to listen for the user input events, handle these events and send them to the game logic layer. The canvas’ other main function is to perform the screen rendering of the game info required by the control layer. This function is requested by the control layer each game iteration, right after the game world update procedure.
- The auxiliary layer responsibility is to provide important information to the whole system, global variables, internationalization resources, and every other support required by several architecture layers.

Usage demonstration

In order to more easily demonstrate the architecture, two simple usage examples are presented below. The first one (Figure 3), shows the architecture behaviour on a scenery where the user is navigating through the game menus (before the game itself). And the second one (Figure 4), shows a game cycle without user intervention.
game cycle, which is responsible for updating and rendering the game progress.

Conclusions

Despite being quite simple, the architecture presented here was used as a base for the development of several games (as for example Atlantis, GoldHunter, SpaceRunner, SeaHunter, PodRace, Stalingrado, X-agon) using different technologies and for different handsets. Some of these games even won wireless game development contests throughout the world, proving their quality. The architecture simplicity is one of its greatest strengths allowing for the simple creation of games letting the developer focus on the game design and on the game rules instead of implementation details (as how the screens will be drawn, or how the events will be handled). Another advantage of such architecture is that it allows task division. Each module can be implemented by a different sub-team after agreeing on a simple set of interfaces between the components.

REFERENCES


AUTHOR BIOGRAPHY

ALEXANDRE DAMASCENO is a Masters Student in AI at Centro de Informática (CIn), Universidade Federal de Pernambuco (UFPE) and has a BSc and is finishing his Masters degree, both in Computer Science.

BÖRJE KARLSSON is a Student Researcher in AI also at CIn/UFPE and has a BSc in Computer Science and is finishing a specialization in Software Engineering at CIn/UFPE.

DANIELLE SILVA is a PHD Student in AI at CIn/UFPE and has a BSc and a Msc in Computer Science.

The three authors are currently working in projects at CIn/UFPE and C.E.S.A.R, related to development and testing for mobile devices.